# *Eventful graphs: the computational complexity of recognizing and realizing properties in changing networks*

DAVID CESARE KUTNER

# Eventful graphs: the computational complexity of recognizing and realizing properties in changing networks

**David C. Kutner**

A thesis presented for the degree of

Doctor of Philosophy

Department of Computer Science

Durham University

United Kingdom

March 2025

# Eventful graphs: the computational complexity of recognizing and realizing properties in changing networks

**David C. Kutner**

**Abstract**

Graph theory and computational complexity are two foundational fields of theoretical computer science. Graphs are excellent models of many real-world systems, but are classically static, whereas the real world changes over time (by design, by nature, or by accident). This thesis explores the computational complexity of problems on so-called *eventful* graphs: that is, graphs which have in some way been extended to express the changeability of the systems they seek to model. A recurring theme throughout this work is the increase in difficulty which results directly from the eventfulness of our models.

The first chapter is an introduction to the thesis, in which we introduce the general themes of the thesis (namely, eventful graphs and computational complexity) and informally describe the subjects of later chapters.

*Payment scheduling in the Interval Debt Model* studies financial networks in which entities are interconnected by debts and the aim is to compute when they should pay one another to optimize some objective (e.g., minimize bankruptcies).

*Temporal Reachability Dominating Sets: contagion in temporal graphs* considers the problem of finding (or preventing) small sets of vertices which collectively reach all other vertices in *temporal graphs*, a well-studied eventful graph model in which edges appear and disappear over time.

In *Reconfigurable Routing in Data Center Networks*, we examine a problem motivated by new technology in interconnection networks, which enables the rewiring of data centers on-the-fly to serve fluctuating demands.

*Detours and Distractions* is an assortment of smaller subchapters on various topics: *Maximal Independent Sets and Boolean Networks* examines a generalization of a classic graph algorithm through the lens of Boolean Networks (also well-studied eventful graphs); *Better Late, then? Delaying connections in temporal graphs* considers a temporal graph problem motivated by train delays; *Partial Domination* delves deeper into some computational complexity questions arising from Chapter 4; and *A nifty Constraint Satisfaction Problem* is dedicated to a proof that a particular combinatorial problem is NP-complete.

Supervisors: Tom Friedetzky, George B. Mertzios, Iain Stewart, and Amitabh Trehan.

# Acknowledgments

First let me thank my supervisors Tom Friedetzky, Iain Stewart, George Mertzios, and Amitabh Trehan for their time, insight, and advice over the course of my studies. Their counsel was invaluable to my development as a researcher and to the writing of this thesis. I would also like to thank my examiners Thomas Erlebach and Andrea Marino for their questions and comments.

I would like to thank my collaborators Laura Larios-Jones, Maximilien Gadouleau, and Anouk Sommer: it was a pleasure and a privilege to work with you, and this thesis is richer for it. I hope our future collaborations are just as enjoyable and fruitful as our past ones!

The inhabitants and regular visitors of MCS-SW2 are also due a mention in these lines: you are the reason I looked forward to coming in to the office for the last three and a half years. Enjoying one's work is a rare privilege, and one which I owe in no small part to you.

I cannot thank my family and close friends enough for their love, their support, and their patience for my combinatorial ramblings. Last, but certainly not least, I would like to thank my wife Emma: you are for me formidable, and you are the reason I looked forward to *leaving* the office.

# Declaration

The work in this thesis is based on research carried out at the Department of Computer Science, Durham University, England. No part of this thesis has been submitted elsewhere for any other degree or qualification, and it is the sole work of the author unless referenced to the contrary in the text.

Some of the work presented in this thesis has been published in journals and conference proceedings - the relevant publications are listed in Section 1.6.

# Contents

# Three Leonards

To make the reading (and writing) of this introduction more pleasant, three Leonards will accompany us through it (and also make the odd appearance later in this thesis). The reader interested only in mathematical content will find the distinctive formatting of these passages useful in skipping them (always as below – with quotations cited as usual, but almost always gleefully disregarding the original context and meaning), and the reader principally seeking whimsy with mathematics interspersed may be better served by other works (e.g., Carroll or Hofstadter). The rather narrow intended audience for these passages is then the reader who is interested in both the content of this thesis and some accompanying whimsy.

---

*Cohen, Da Vinci, and Euler meet in Manhattan. Realizing himself the host-est of the unlikely trio, Cohen unfolds a map to explain their whereabouts.*

*C. : New York is cold, but I like where I'm living [1, Famous Blue Raincoat]*

*E. : In America! [2, Letter XV, p. 61]*

*D. : So these are all rivers?*

*C. : We call them rivers – Hudson, Harlem, East – that last one is saltwater.*

*D.* (thoughtfully)*: Between river and river... [3, Note 1092] The greatest river in our world is the Mediterranean river, which moves from the sources of the Nile to the Western ocean. [3, Note 1092]*

*E. : The ocean in many places has such rapid currents that it resembles a running river. [4, Letter XXX, p. 206] But in any case, if those are rivers, then* (with visible excitement) *these must be bridges?*

D. makes a sketch of the map in his notebook.

*D. : I should like to see these bridges!*

*E. : Me as well! Perhaps we could do so in one trip, never using the same crossing twice?*

*C. : Maybe... But then you should keep note of what each crossing is – firstly, those are tunnels* (D. and E.'s enthusiasm for them palpably diminishes)*, this one is a rail bridge, that one changes direction twice a day, these ones you'd have to cross with the subway, but the view makes up for the olfactory experience ...*

Some time later, E. has drawn a cleaner version of C.'s sketch (omitting landmass shapes) and the trio has agreed on an itinerary.

*C.* (walking behind)*: "Follow me," the wise man said. [1, Teachers]*

---

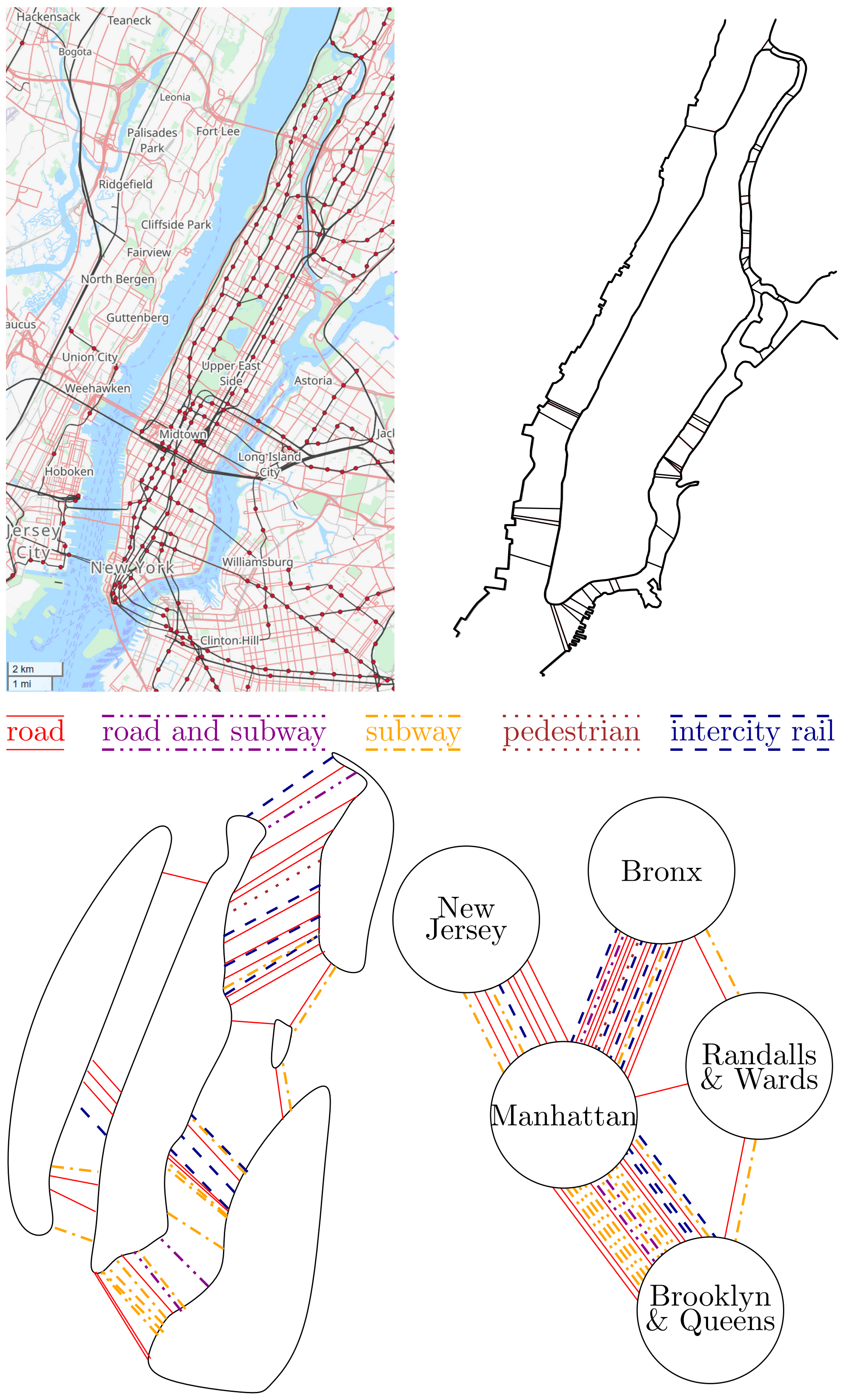


Figure 0.1: Abstracting away the details of Manhattan's crossings. Top left map [5]; top right map [6].

Chapter 1

# Introduction

## 1.1 What this thesis is about

This thesis sits at the intersection of "eventful" graph theory and computational complexity. We first introduce these two recurring concepts at a high level, and then go into a little more detail about each of the forthcoming chapters.

### 1.1.1 (Eventful) graphs

A graph is a computer scientist's[1] second-favorite way of representing the New York subway. It abstracts away all the unnecessary details - the physical distance between stops, whether a crossing is a bridge or a tunnel, the names of the trains, and so on. Only the stations and their interconnections remain – a result not dissimilar to a subway map.

Graphs classically consist of vertices, edges, and nothing more. For example, Figure 1.1 shows a small graph $G$ on five vertices and six edges. The manner in which a graph is drawn is unimportant - the object *is* the vertices and their connections, *not* their positions. Two graphs $G$ and $H$ which can be drawn in the same shape are called *isomorphic* because mathematicians enjoy using Greek (literally "iso", equal, and "morphic", shaped). Figure 1.2 shows two other drawings of $G$, and two drawings of graphs isomorphic to $G$. In this work we will often say that $G$ and $H$ are the same graph, rather than isomorphic to one another, but the distinction occasionally matters.

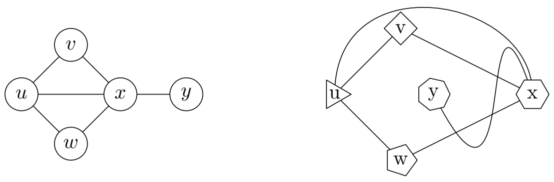


Figure 1.1: Two drawings of the same small graph $G = (V, E)$, with $V = \{u, v, w, x, y\}$ and $E = \{(u, v), (u, w), (u, x), (v, x), (w, x), (x, y)\}$. The manner in which vertices and edges are drawn, like the fonts, are immaterial – only the relationship between vertices and edges matter.

[1] namely, the author

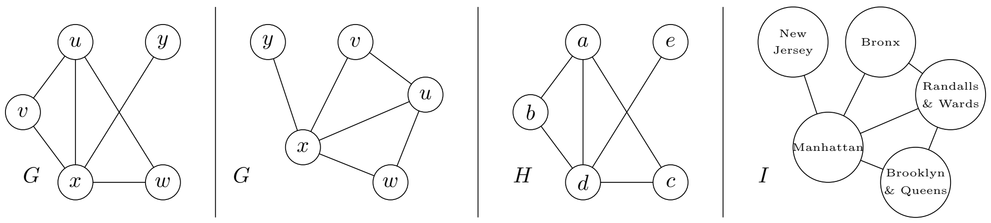


Figure 1.2: Two more drawings of the graph $G$ from Figure 1.1, and two graphs $H$ and $I$ which are isomorphic to $G$.

Graphs are relevant far beyond rail transport, of course. Myriad real-world systems – including many things which aren't networks themselves – can be modeled using graphs. Although graphs make for beautiful theory and fascinating structure, they are static, and many of the systems we actually wish to study *change*. In other words, the real world is *eventful*. We shall use the umbrella term *eventful graph* to designate a graph which is somehow extended to account for the changeability intrinsic in real-world systems. The nature of the events in question is, of course, very relevant to the exact model applied; for example, these may be:

- changes which may be chosen (e.g., when to pay a debt, or how to wire a datacenter)
- locally predictable occurrences beyond our control (e.g., the spread of a fire, disease, or information)
- scheduled events (e.g., trains) – which may themselves be disrupted in unforeseen ways.

(A real system may change in many different ways, of course - in which case we might try to model several kinds of changes simultaneously, or instead focus only on the most frequent culprit for the system's dynamism.)

One example of an eventful graph (and, incidentally, the author's favorite way to model the New York subway) is a *temporal graph*. A temporal graph can be thought of as a static graph with a schedule. Like a static graph, it has vertices and connections, but the connections appear and disappear over time. Figure 1.3 shows a small temporal graph $\mathcal{G}$ and *snapshots* $G_t$ of $\mathcal{G}$ at different times $t$. We say that a vertex *reaches* another vertex if there is a time-respecting path from the former to the latter. For example, in Figure 1.3, $u$ reaches $y$ through a path which goes from $u$ to $x$ at time 1, then from $x$ to $y$ at time 2, and $y$ reaches $u$ through a path which goes from $y$ to $x$ at time 2, then to $w$ at time 3 and finally to $u$ at time 4. On the other hand, $v$ does not reach $y$ at all - because no edges leaving $v$ are active early enough to enable a connection with the only edge into $y$, which is at time 2.

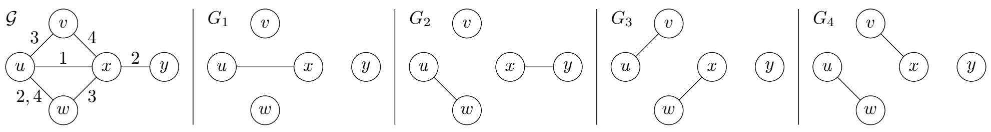


Figure 1.3: A small temporal graph.

### 1.1.2 Computational Hardness: what is as hard as Sudoku?

*Graph theory*'s (much younger) companion throughout this work is *computational complexity theory.*

---

*D: It is a useful companionship, for one is not good for much without the other. [3]*

---

A classic problem known to casual puzzlers and devout combinatoricists alike is Sudoku: given a grid, usually of $9 \times 9$ cells (some empty, some not), the object is to write a single digit (the eponymous "su doku") in each empty cell so that each row, column, and $3 \times 3$ subgrid contains exactly one of each digit between 1 and 9. Sudoku is closely related to Latin Squares – which, like graphs, are combinatorial objects of Eulerian (re-)invention [7].

---

The Leonards sit at a café near Union Square Park, huddled around the final pages of a newspaper.

*D.* (pointing)*: Put before you three or four. [3, Note 798]*

*E. : Yes, in the fourth row, fifth column - right there!*

*C.* (noting the numbers on the grid in the newspaper)*: It goes like this: the fourth, the fifth? [1, Hallelujah]*

Later, at the same table, the Leonards are puzzled.

*D. : Well this is tricky. It must be three or four, and yet it cannot be either of them.*

*C. : We must have made a mistake earlier.*

*E. : It cannot be affirmed. [2, Letter XVII p.70]*

*C. : You mean there might be a mistake in the puzzle itself?*

*E. : It must be considered [4, Letter IV p. 92].*

*C. : Fine – EITHER there is a mistake in the puzzle, OR, we made a mistake earlier.*

*E. : It cannot be denied [4, Letter LVI p. 39].*

*C. : Well, I think the puzzle is wrong.*

*D. : Well, we can call the newspaper –*

*C. : And who shall I say is calling? [1, Who by Fire] The ghost of you and me? [1, Is This What You Wanted]*

*D. : – unimportant; if it is valid, they should have a solution for it, right? And then we would be able to check it ourselves, so that-*

*E. : -it cannot be doubted [4, Letter LVII p. 43].*

---

It is generally accepted that Sudoku makes for challenging puzzles[citation needed]. It is also intuitive checking whether a particular solution to a puzzle is correct is easy: it suffices to verify column by column, row by row, and subgrid by subgrid that all symbols appear exactly once. Of course, with modern computers, a standard $9 \times 9$ Sudoku puzzle can be solved quite easily [8]. This is not true for *generalized* Sudoku (as illustrated in Figure 1.4). Indeed, if a program which efficiently[2] solves Sudoku

[2]Explained more formally in the sequel.

on an arbitrary-sized board were proven to exist, this would entail that P=NP and thus solve one of the major open questions in mathematics.

P and NP are sets of *decision problems*. A "problem" as computer scientists understand the word is essentially a function – a mapping from inputs to outputs. An input of a problem is called a "problem instance" and the output is then the answer for that particular instance. If the answer sought is a simple "yes" or "no" (e.g., "Is the given Sudoku puzzle valid?"), we call the problem a *decision problem*, and if the answer sought is not binary (e.g., "Given a Sudoku puzzle, return the solved puzzle.") then we call the problem a *function problem*.

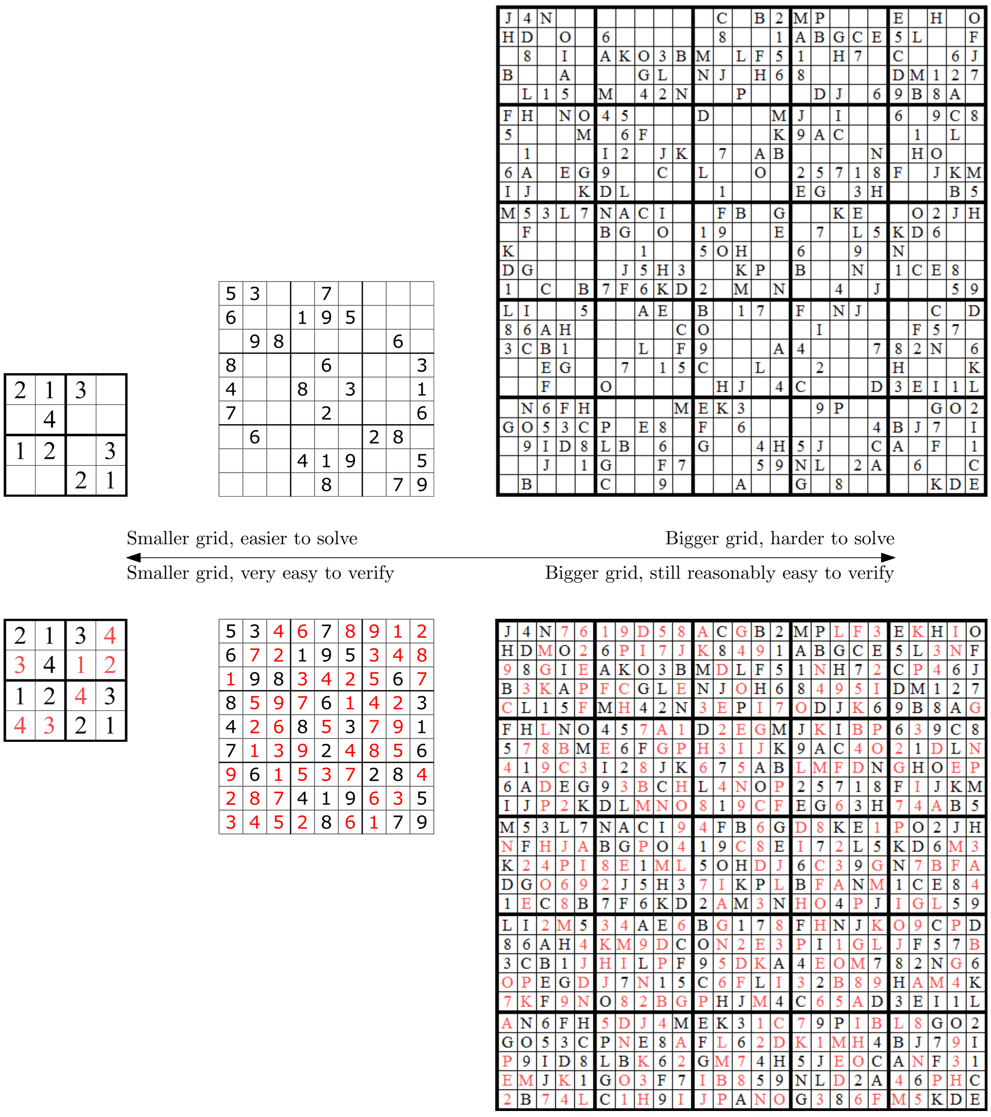

Smaller grid, easier to solve
Bigger grid, harder to solve
Smaller grid, very easy to verify
Bigger grid, still reasonably easy to verify

Figure 1.4: Sudoku grids of varying sizes and their solutions (puzzles generated with [8]).

For example, on the top half of Figure 1.4 are three instances to the decision problem SUDOKU[3] (which asks if the given Sudoku puzzle is valid). In this case, the answer to the problem is YES for all three instances, (we call them yes-instances). We note at this point that verifying that the answer to any yes-instance of SUDOKU is indeed YES is straightforward – if the solution is given. On the other hand, it is

[3]Computer scientists usually present decision problems in small capitals, as shown here. Some (but certainly not most) extend this idea to their speech, shouting the names of decision problems.

not immediately clear whether all no-instances have some equivalently easy to verify proof.

Central to the field of computational complexity is the question of what resources are necessary (or sufficient) to solve (or verify answers to) different problems. Specifically, we are interested in how the resource requirement to solve some problem *grows with the size of the input*. Most often, the resource which is of interest is time.

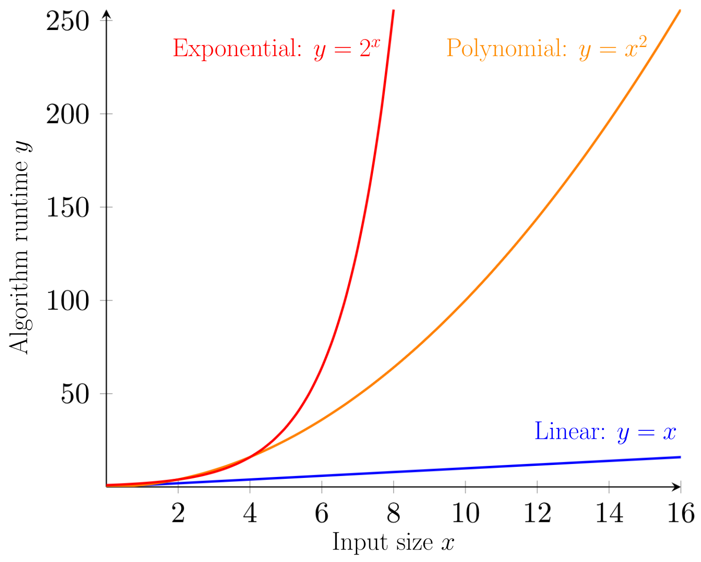


Figure 1.5: Illustration of different types of functions' growth rates.

If the amount of time needed to solve (or, respectively, verify) the problem doubles any time the input size doubles, we say that the problem is solvable (resp. verifiable) in *linear time*, and if the amount of time needed increases by the same factor $k$ anytime the input size doubles, then we say the problem is solvable (resp. verifiable) in *polynomial* time. We generally consider that algorithms which run in polynomial time are practical, and that ones which require more than polynomial (e.g., exponential) time are not. (Linear, polynomial, and exponential growth are illustrated in Figure 1.5.)

P and NP are *classes of decision problems*: P contains all those decision problems which can be *solved* in polynomial time, and NP contains all those decision problems whose *solutions can be verified* in polynomial time. Note that every decision problem which can be solved in polynomial time can also be verified in polynomial time (simply solve the problem from scratch - no proof is needed!) but the converse is not necessarily true. Therefore, P is a subset of NP. Though it is widely believed that P and NP are different, a proof remains elusive. More specifically, it is known that if P and NP are indeed different, then a particular problem called SATISFIABILITY is in NP but not in P.

Let us return to the example of SUDOKU. We would like to know whether there could be a polynomial-time algorithm for the problem - we suspect this is not the case, and would like to prove it. We know that SUDOKU is verifiable in polynomial time, so it is in NP. If we were able to prove that SUDOKU is *not* solvable in polynomial time, this would entail that P≠NP - perhaps too ambitious a goal. Enter *reductions*: transformations which “translate” between problems, turning instances of a problem

$\Gamma$ into instances of another problem $\Pi$, so that yes-instances are mapped to yes-instances and no-instances are mapped to no-instances. We denote by $\Gamma \leq_{\text{poly}} \Pi$ the existence of polynomial-time reduction from a problem $\Gamma$ to a problem $\Pi$. This is a proof that $\Gamma$ is "at most as hard" as $\Pi$ – since the existence of a polynomial-time algorithm $\mathcal{A}$ for $\Pi$ entails the existence of a polynomial-time algorithm for $\Gamma$ by first applying the reduction and then solving the obtained instance of $\Pi$ using $\mathcal{A}$. This idea is illustrated in Figure 1.6. In our case, there is a reduction from SATISFIABILITY to SUDOKU [9] – so there can be no polynomial-time algorithm for solving SUDOKU unless P=NP.

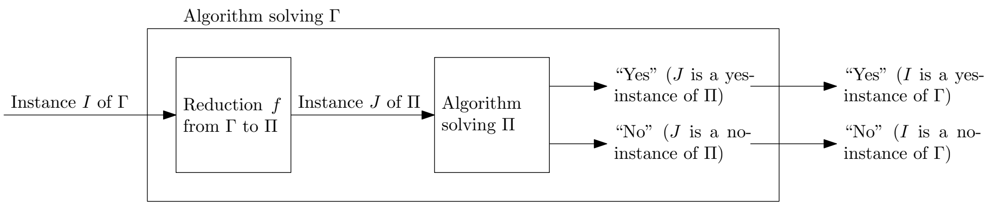

Algorithm solving $\Gamma$
Instance $I$ of $\Gamma$
Reduction $f$ from $\Gamma$ to $\Pi$
Instance $J$ of $\Pi$
Algorithm solving $\Pi$
"Yes" ($J$ is a yes-instance of $\Pi$)
"No" ($J$ is a no-instance of $\Pi$)
"Yes" ($I$ is a yes-instance of $\Gamma$)
"No" ($I$ is a no-instance of $\Gamma$)

Figure 1.6: A polynomial-time reduction from a problem $\Gamma$ to a problem $\Pi$ entails that if we have an algorithm solving $\Pi$ in polynomial time, then we get an algorithm solving $\Gamma$ in polynomial time.

We can extend this idea to other problems: GRAPH COLORING asks if a given graph can be colored "properly" with some number of colors, i.e. whether all vertices of the graph can be assigned a color so that no two neighboring vertices are assigned the same color. PRECOLORING EXTENSION asks the same, but the graph we are given is *precolored* – some of the vertices already have a color assigned to them. These concepts are illustrated in Figure 1.7.

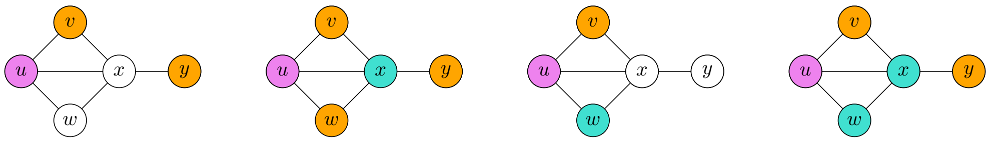


Figure 1.7: From left to right: a precoloring of $G$; a proper 3-coloring of $G$ extending the precoloring; a second precoloring of $G$ (which cannot be extended to a proper 3-coloring); an extension of the second precoloring (which is not proper: $w$ and $x$ share a color).

A helpful property of polynomial-time reductions is that they can be composed, and so polynomial reducibility is transitive: if $\Gamma \leq_{\text{poly}} \Pi$ and $\Pi \leq_{\text{poly}} \Lambda$ then $\Gamma \leq_{\text{poly}} \Lambda$. This means that if we are interested in showing some new problem (say, GRAPH COLORING) is as hard as SATISFIABILITY, we can do this by reducing from SUDOKU (because we already know SATISFIABILITY $\leq_{\text{poly}}$ SUDOKU)[4].

For example, SATISFIABILITY is reducible in polynomial time to SUDOKU, and SUDOKU is reducible in polynomial time to GRAPH COLORING (see Figure 1.8), so SATISFIABILITY is reducible in polynomial time to GRAPH COLORING. We can imagine all decision problems as being elements of an enormous set which is (partially) ordered by these polynomial-time reductions. A problem $\Pi$ is *complete* for a set if it is in the set, and every other problem in the set is reducible to $\Pi$ - so $\Pi$ is as

[4]We could also use Karp's proof that SATISFIABILITY $\leq_{\text{poly}}$ GRAPH COLORING from 1972 [10], but this would be less fun as an example because his proof does not use SUDOKU (probably because Sudoku didn't exist yet [7]).

hard as anything else in the set. It is known that SATISFIABILITY is complete for NP [10]. So, because SATISFIABILITY $\leq_{\text{poly}}$ SUDOKU, we know that SUDOKU is also complete for NP – since it is in NP, and every other problem in NP can be reduced to SUDOKU in polynomial time (via SATISFIABILITY). Figure 1.9 shows reducibility among some classic combinatorial problems. Note that some problems in NP are not in P and are also not NP-complete (unless P=NP, in which case everything collapses) – the most famous candidate for such a so-called *NP-intermediate* problem is GRAPH ISOMORPHISM.

Much of this thesis is dedicated to proving that problems which we know are in NP are in fact NP-complete[5], or, conversely, dedicated to finding polynomial-time algorithms for (restrictions of) these problems. NP-intermediate problems also make an appearance in Chapter 5c.

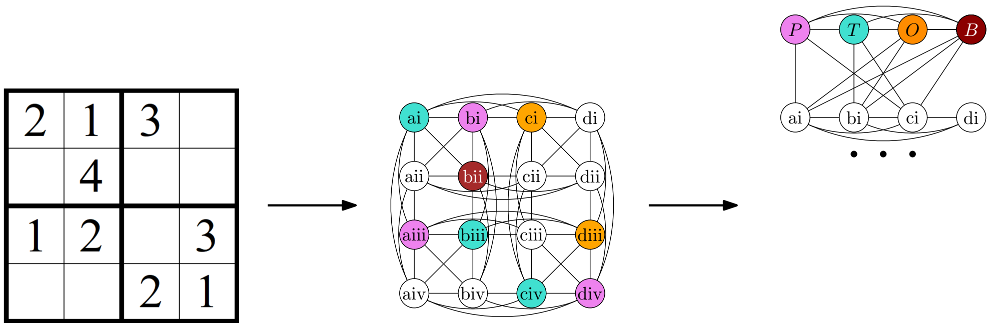


Figure 1.8: Reduction from SUDOKU to PRECOLORING EXTENSION to GRAPH COLORING. First, the Sudoku grid on the left is valid if and only if the precoloring of the graph in the center can be extended to a proper 4-coloring – the numbers 1-4 correspond to the 4 colors (pink, turquoise, orange, and brown) used in the precoloring, and the restrictions on placing numbers in the Sudoku grid exactly correspond to the edges in the graph shown in the center.
Second, the precoloring of the graph in the center can be extended to a proper 4-coloring if and only if the graph on the right has a proper 4-coloring at all: the special *palette* vertices $P, T, O, B$ at the top of the figure must be colored differently (since they are all connected), and we can assume that $P$ is pink, $T$ is turquoise, $O$ is orange, and $B$ is brown in a proper 4-coloring. Vertices which are precolored in the center graph (corresponding to filled-in cells in the grid on the left) are connected to all the palette vertices of a different color from their precoloring (e.g., ci is connected to $P, T$, and $B$, but not to $O$, forcing it to be colored orange). Vertices which are blank in the center graph (corresponding to blank cells in the grid on the left) are not connected to any palette vertices.

Faced with a hard problem, theoretical computer scientists often restrict the class of instances considered. For example, now that we know SUDOKU is NP-complete, we might be interested in whether it is computationally hard to solve SUDOKU on specific types of grids. One possibility is to restrict the structure of the instance – for example, it is clear that if each row and column in a Sudoku grid has at most one blank, we can easily fill in all these blanks (and check that the solution obtained this way is correct).

A more granular approach is *parameterization*: we identify a measure of instances which, if limited to *any* constant, allows us to solve the problem in question in polynomial time. We again turn to SUDOKU for an example. Intuitively, when few cells

[5]Meaning we cannot hope to solve it in polynomial time unless P=NP.

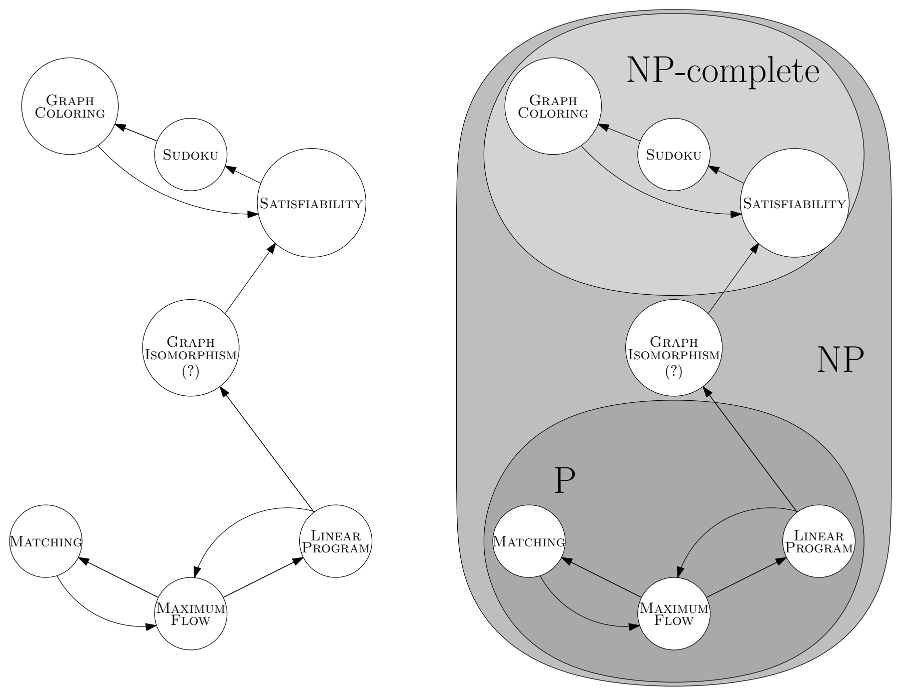


Figure 1.9: Reducibility among problems in P and NP. An arrow from Γ to Π means $\Gamma \leq_{\text{poly}} \Pi$. Reductions implied by transitivity are not shown.

are empty, it is easy to solve a Sudoku grid – and hence to check whether it has a solution. This can be formalized: there exists an algorithm which, given a Sudoku grid of size $n$ with $k$ many blank cells, finds a solution in $O(k^k \cdot n)$ time. A parameterized problem which can be solved by such an algorithm is called *fixed-parameter tractable* (fpt), because if the parameter $k$ is set to be any fixed constant (be it 3 or 1000), the algorithm's running time is polynomial in the size of the input.

## 1.2 The Interval Debt Model

The Leonards overhear a conversation at the next table.

*Alice: Heya Bob! To help us figure things out, I asked my accountant to meet us here, she should be –*

*Bob: Accountant? You only owe me two dollars! You've got the whole week to come up with the money! How much are you paying her to figure this stuff out for you?*

*Alice: Six dollars, but I don't see what that has to do with anything. Ah, here she is! This is Carol, I don't know if you've met.*

*Carol: We have!*

*Bob: She's my accountant too. Which reminds me, I also owe her six dollars. We said by the end of next week, right?*

*Carol: Indeed!*

*Alice: He gets till next week? You gave me two days!*

*Carol* (shrugging)*: You're a higher-risk client. To be honest, I don't see any way you come out of this without filing for bankruptcy.*

...

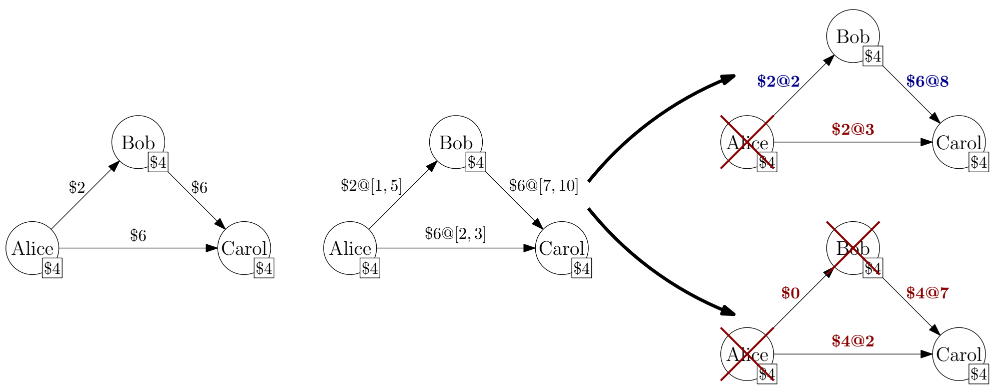


Figure 1.10: A small financial network. Left: Alice, Bob and Carol each have $4; Alice owes Bob $2, and Alice and Bob each owe Carol $6. Center: the *time* each debt is due is included (e.g., Alice owes Bob $2 *which must be paid between the 1st and 5th of the month*). Right: two possible *schedulings* of the payments among the trio – one of these is "good", resulting in only one bankruptcy and one underpaid debt, and the other is "bad", resulting in two bankruptcies and three underpaid debts.

Chapter 2 is about *financial networks*: financial entities interlinked by debts between them (modeled as a *directed graph*, because debts are not symmetric relationships). Theoretical works on financial networks study formalizations of this system, usually centered on some form of the payment question: *Who pays whom, and how much?* (We extend this with *"and when?"*, though most other works do not.)

Previous studies have considered both computational complexity questions: *How hard is it to find a ("good") answer to the payment question?* [11, 12] and game theory questions: *If our vertices are selfish agents, then what strategy will each node use, and what answer to the payment question do these strategies produce?* [13]. Some of these works also consider *bailouts* as possible payments: direct cash injections into specific nodes in the system. The seminal work in this area is Eisenberg and Noe's [11]. Their model has been extended in many ways, though these are seldom eventful (in the sense which we set out in Section 1.1.1). A notable exception is Papp and Wattenhofer's study of sequential defaulting in financial networks [14], where the order in which bankruptcies are announced is highly relevant to the ultimate outcome. At the core of that model are *Credit Default Swaps* (CDSs), that is, financial derivatives which effectively allow entities to bet on one another's bankruptcies. That work inspired us to consider the complexity of an eventful model of financial networks which had only simple debts, and no CDSs, but left entities with some latitude as to when payments should be made.

The model we consider in Chapter 2 incorporates a temporal dimension to our questions. Looking at the small network in Figure 1.10, the question of *when* different parties can (or do) pay one another has a direct effect on possible outcomes. For example, if Alice's debt to Bob was due strictly after her debt to Carol (say, in the interval $[4, 6]$) then it would be impossible for Bob to avoid bankruptcy (as in the "good" scheduling in the top right). Conversely, if Alice's debt to Carol was due strictly after her debt to Bob (say, in the interval $[6, 10]$) then there would not be any possible scheduling in which Bob is left bankrupt (as in the "bad" scheduling in the bottom right).

The *Interval Debt Model* (IDM) is our contribution. The questions we consider are all about the aforementioned payment scheduling:

- Bankruptcy Minimization asks if there is a schedule with at most some number of bankruptcies $k$, and Perfect Scheduling asks if it is possible to schedule payments so that *no one* goes bankrupt,
- Banrkuptcy Maximization asks if there is a schedule with at *least* some number of bankruptcies $k$ (because the Bobs of the world are anxious to know what happens to them in the worst-case scenario),
- Bailout Minimization asks how small a cash injection is sufficient to make a perfect schedule (with no bankruptcies) possible.

In Chapter 2, we show that all of these problems are NP-complete, in many cases even on very restricted input instances. On the other hand, we show that Perfect Scheduling is in P for nicely-structured instances (specifically, when debts are all directed "away" from someone), and also when we consider a model in which fractional payments are allowed (so, e.g., Bob can pay Carol three sevenths of a dollar on Monday and the rest of his debt on Tuesday).

## 1.3 TaRDiS

*The Leonards are in Times Square and have a run-in with The Doctor.*

*C: Where's our Swiss friend? Ah, here he is!*

*E* (coughing)*: I had the strangest encounter – a police box just coughed on me!*

(C and D also start coughing.)

*Dr. : Very sorry gentlemen, it seems you are infected with my TARDIS.*

*C: Your what?*

*Dr. : My TARDIS – big blue box, bigger on the inside, says "Police" on the front, occasionally becomes a viral infection?*

*C* coughing*: Everybody knows that the plague is coming. [1, Everybody Knows]*

*Dr. : Indeed – the TARDIS is exploding right now [15, S5E12].*

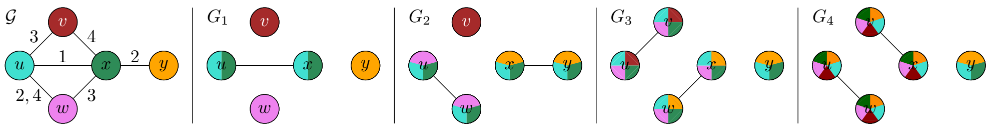


Figure 1.11: Contagion in a temporal graph. A vertex in snapshot $t \in \{1, 2, 3, 4\}$ is drawn with its own color, and that of any which can reach it by time $t$.

Chapter 3 is about contagion in temporal graphs, and in particular about *Temporal Reachability Dominating Sets* (or, more briefly, *TaRDiSes*). A *dominating set* in a (static) graph is a set of vertices which "sees" every vertex (illustrated in Figure 1.12), and a TaRDiS is a set of vertices that reaches every vertex in the graph through a temporal path. Figure 1.11 illustrates spread in the small temporal graph from Figure 1.3.

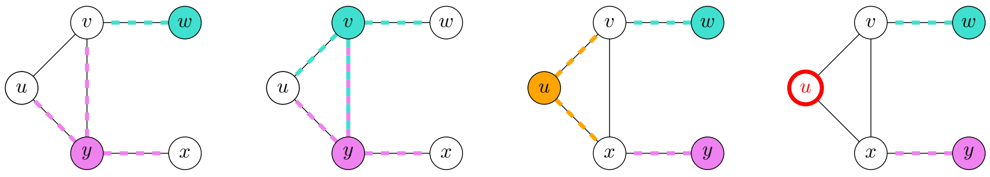


Figure 1.12: Different sets of vertices in a small graph, showing "lines of sight" of each vertex in each set as dashed lines of the appropriate color. $\{w, x\}$, $\{v, y\}$ and $\{u, w, y\}$ are each dominating sets, whereas $\{w, y\}$ is not, because $u$ is neither in the set nor adjacent to one of $w$ and $y$.

We note that temporal paths (unlike static paths) cannot necessarily be followed in reverse: a path from $u$ to $v$ does not entail a path from $v$ to $u$, even if each connection is symmetric.

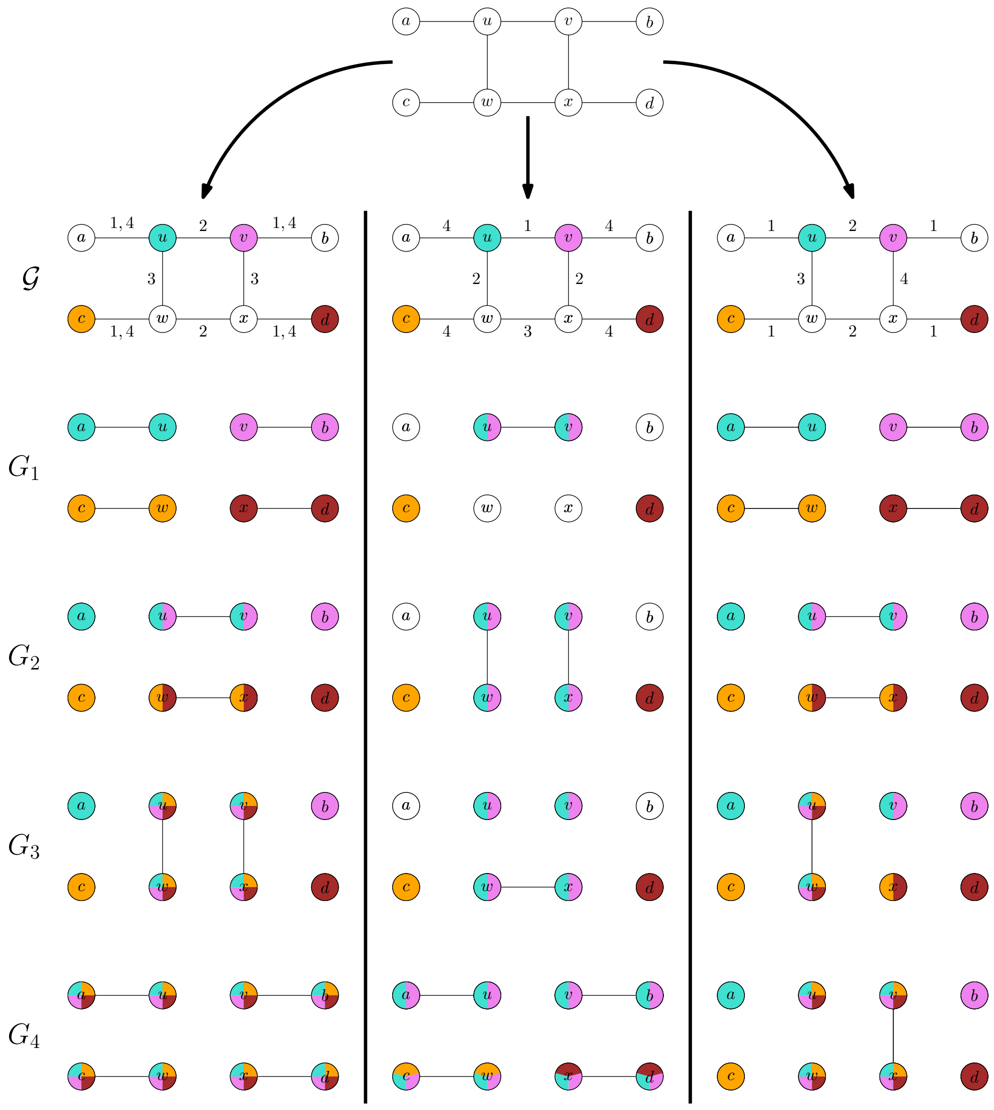


Figure 1.13: Spread from the vertices $\{u, v, c, d\}$ in three different temporizations of the same footprint graph. On the left, any single vertex reaches all other vertices in the graph; in the center, either $u$ or $v$ suffice to reach all vertices; and on the right, a set of four vertices is necessary to reach the entire graph.

The temporal graph community has considered a large variety of reachability problems. These have been studied in the context of network design [16, 17, 18] and transport logistics [19, 20] (where connectivity and reachability are considered positive), and the study of epidemics [21, 22, 23, 24] and malware spread [25] (where they are not). Another part of the literature on temporal graphs focuses on so-called *temporization* problems, in which a static graph (called the *footprint*) is given and the task is to find whether it is possible to schedule its edges to achieve some temporal property. One classic example of such a property is *connectedness*, i.e., every vertex having a path to every other vertex [26]; other properties (also with various constraints for the schedules which may be allowed) have also been explored [27, 28, 29]. Different temporizations of the same footprint graph may have very different reachability properties, as illustrated in Figure 1.13.

The intuition behind TaRDiS is that in a well-connected temporal graph, we would expect that a small number of vertices can (collectively) reach the entire graph. The TaRDiS problem asks, given a temporal graph $\mathcal{G}$ and an integer $k$, whether there is a TaRDiS in $\mathcal{G}$ of size at most $k$.

We also consider the MaxMinTaRDiS problem, a temporization problem where the desired property is that there is no small TaRDiS (i.e., the aim is to **max**imize the size of the **min**imum TaRDiS). Intuitively, MaxMinTaRDiS asks: "How badly-connected could *any* temporal graph with this footprint be?" For example, if $G$ is the graph at the top of Figure 1.13, then $(G, 4)$ is a yes-instance of MaxMinTaRDiS, because the temporization shown on the right-hand side of the figure has a minimum TaRDiS of size (at least) 4.

Our contributions constitute a near-comprehensive study of the complexity of both problems, in different settings and under different restrictions. Without introducing too much technical detail, the principal results are the following:

- NP-completeness (or worse[6]) of TaRDiS and MaxMinTaRDiS even when the lifetime (total number of snapshots) of the temporal graph is limited to three.
- A proof that one particular variant of MaxMinTaRDiS is exactly the (already known and studied) static graph problem Distance-3 Independent Set.
- On the parameterized side, fpt algorithms for both problems using parameters combining structural properties of the footprint with the lifetime of the temporal graph (or, in the case of MaxMinTaRDiS, lifetime of the desired temporization).

[6]We show $\Sigma_2^P$-completeness of a specific variant of MaxMinTaRDiS; the practical implications of this result are that we cannot even hope to *verify* yes-instances of this variant of the problem in polynomial time (unless some common assumptions fail – in this case the Polynomial Hierarchy collapsing to the first level).

## 1.4 Wiring datacenters

---

The Leonards have just finished going through the Lincoln Tunnel(s) three times on their tour of the city's crossings.

*E. : Well that was. . . gray.*

*D. : So unpleasing a sight. [3, Note 1288]*

*E. : So because it's – what did you call it? – "rush hour", the central tunnel is leading from that side to this one?*

*C. : All through the evening. [1, Famous Blue Raincoat]*

*D. : Do you suppose we could change not just the direction, but also the endpoints? Yes – a floating bridge of sorts! Then it could lead from any landmass to any other!*

A ferry cruises past them.

*E. : Supposing we had your flotilla of bridges, we would still need to worry about* how *to arrange them. . .*

---

Chapter 4 is about choosing a "good" configuration for a graph, some edges of which can be changed, based on the tasks it needs to perform. The specific motivation for this research is found in huge facilities full of interconnected computers: *data centers*. A data center consists of a large number of computers (servers), which traditionally would be connected to one another with physical copper cables (or optic fibers). A relatively recent development is the viability of so-called free-space optics in this context[7]: by mounting a mirror on the ceiling, and laser terminals on top of server racks, it is possible to create optical connections between any pair of servers in a room. Free-space optics are also being developed at a much larger scale, as shown in Figure 1.14.

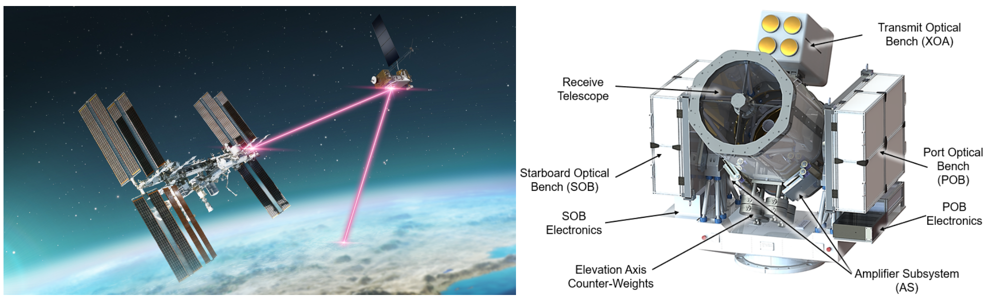


Figure 1.14: Left [31]: NASA's ILLUMA-T system on the International Space Station communicates via a satellite, *Laser Communications Relay Demonstration* (analogous to our ceiling mirror). Right [32]: NASA's *Low Cost Optical Terminal*.

The advantage of this approach is that it allows for rapid reconfiguration; the top-of-rack terminal can be re-oriented in a matter of milliseconds to nanoseconds [33, 34, 35], which means that by having just a single laser terminal on each server rack it is possible to create a direct optical connection between any pair of servers as and when it is needed. Of course, it is possible to have several terminals on top of each rack, which would enable several simultaneous optical connections. We are more

[7]Free-space optics in other contexts have been in use for much longer [30].

specifically interested in *hybrid* networks, which combine a reconfigurable network architecture with a traditional fixed data center network. Information that needs to be sent from one server to another may then be routed either in the fixed cables of the network, through the free-space optical links, or through a combination of the two.

Hybrid networks can adapt in near-real time to demands, but how hard is it to choose the *right* adaptation for some given demands? This is the Reconfigurable Routing Problem (RRP). Slightly more formally, in this problem: the ceiling mirror is called a *reconfigurable switch*; a free-space optical link (server-to-server, including a "mirror bounce" through the reconfigurable switch) is called a *dynamic link*; and our copper cables are called *static links*.

Previous work on this problem [36, 37, 38] established that it is NP-complete, and identified some interesting parameters for the problem, such as the number of terminals on each server rack: $\Delta_S$, the *switch degree*, controls how many connections each server rack may make to the optic switch, and $\sigma$, the *segregation parameter*, controls how many times a packet being routed through the network may switch mediums (e.g., a dynamic-static-dynamic path would be disallowed if $\sigma$ were set to 1 or less). Similarly, the *dynamic link limit* $\delta$ prescribes the maximum number of dynamic links any packet may be routed through (so if $\delta = 1$ then a dynamic-static-dynamic path is disallowed but a static-dynamic-static path is allowed). The results shown in these works largely would not hold for highly structured networks. However, real-world data centers are often just that: highly structured [39, 40].

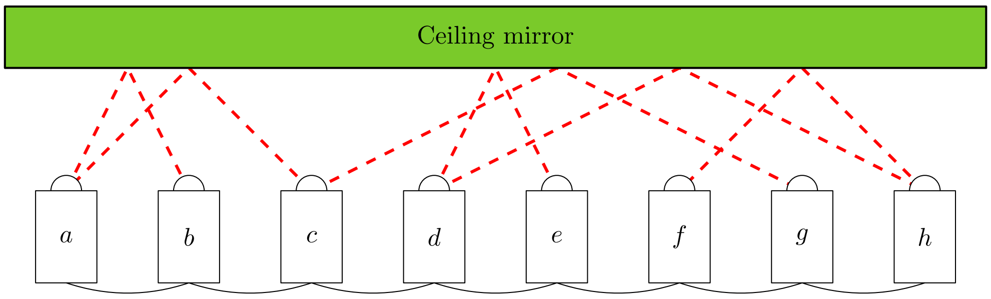


Figure 1.15: A diagram of a small (8-rack) data center with a physical cable connecting all racks in series, and a ceiling mirror enabling free-space optical connections between racks. In the configuration shown, rack $b$ may communicate with rack $c$ either through a single copper link or through two mirror bounces (through rack $a$'s terminal).

Consequently, we consider restrictions of RRP to instances where the network has "realistic" properties. In particular, we expect:

**Symmetry:** each server has identical hardware, and the "view" of the network from each server is the same

**High connectedness:** the network stays connected even if some cables are damaged

**Low diameter:** for any two servers, there is a "short" path connecting them

**Low degree:** each server has "few" direct connections to other servers

The problem $\Delta_S$-Switched RRP is the restriction of the problem RRP to those instances where every server rack has exactly $\Delta_S$ terminals. Figure 1.15, for example, shows an instance of 2-Switched RRP, and not of 1-Switched RRP, since $a$ is simultaneously maintaining optic links to both $b$ and $c$. Our main findings are:

- 2-Switched RRP and 3-Switched RRP are NP-complete regardless of the topology of the static part of the network.
- 1-Switched RRP with $\sigma$ set to 0 is in P, and 1-Switched RRP is in P if we restrict ourselves to instances where every pair of servers is connected directly by a link.
- 1-Switched RRP is NP-complete when the static portion of the network is a hypercube (a highly structured graph on which the BCube network architecture is based [40]), even when either $\sigma = 3$ or $\delta = 1$.

## 1.5 Detours and distractions

Chapter 5 presents several different strands of research which we wished to include but which did not each warrant a separate chapter.

### 1.5a Boolean Networks

---

*The Leonards gather by The Little Red Lighthouse. A large lever protrudes from the structure, angled down towards a plaque which reads OFF. Across the water is another small lighthouse, its own lantern lit.*

*C: Can you make out what the sign on the front of that lighthouse reads?*

*E: My eyesight isn't what it once was...*

*D: Perhaps if we had a looking glass, or more light from our side?*

D. flips the lever to ON. The Little Red Lighthouse whirrs with electricity as its beacon beams across the Hudson. When it reaches the small lighthouse on the other bank, that one's lantern goes dark. The Leonards clearly make out the sign on the front: “Chez Boole”.

*D: Do you think we vexed them? Perhaps shouldn't have – I'll turn it back off.* He flips the lever back to OFF.

After a beat, the lighthouse across the water comes back on.

---

Chapter 5a is about a type of eventful graph called a *Boolean network* (which we will explain shortly), and *maximal independent sets*.

An *independent set* in a graph is a set of vertices which shares no edges, and a *maximal independent set* (MIS) is an independent set to which no vertex can be added. Maximal independent sets should not be confused with maxim**um** independent sets, which are those independent sets of maximum size for the graph. Figure 1.16 illustrates the differences between these concepts. Maximal independent sets are fundamental objects in graph theory, and there has been a rich line of study into how different distributed computing models can compute maximal independent sets [41, 42, 43].

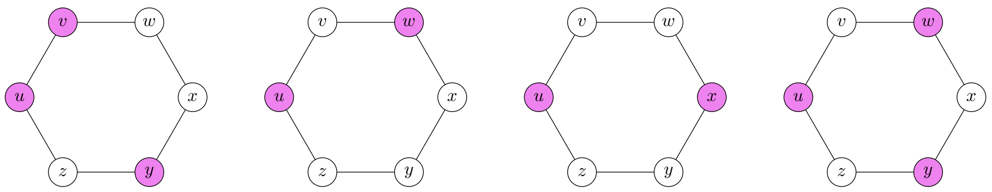


Figure 1.16: In the cycle graph $C_6$ with vertices $\{u, v, w, x, y, z\}$. From left to right: $\{u, v, y\}$ is not an independent set, because $u$ and $v$ share an edge; $\{u, w\}$ is not a maximal (and hence not a maximum) independent set, because $y$ could be added to make a bigger independent set; $\{u, x\}$ is a maximal independent set (no vertex can be added), but is not a maximum independent set (because there exists a larger one); $\{u, w, y\}$ is a maximum (and hence maximal) independent set.

A simple greedy algorithm to find a maximal independent set (MIS) in a graph starts with the empty set and visits every vertex, adding it to the set if and only if none of its neighbors are already in the set. In this chapter, we consider (the complexity of decision problems related to) the generalization of this MIS algorithm wherein any starting set is allowed. Two main approaches are leveraged: Boolean networks (which predate our work and ourselves), and the notion of a constituency in a graph (which we introduced).

A constituency of a graph is a set of vertices that is dominated by an independent set which is outside the constituency (in the DOMINATING SET sense). Recognizing a constituency is NP-complete, a fact we leverage repeatedly in our investigation. Figure 1.17 illustrates different (non-)constituencies in a heptagon graph.

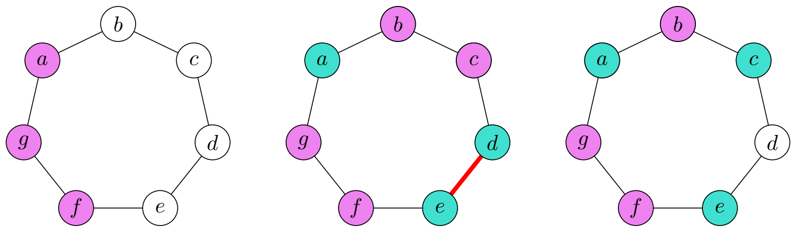


Figure 1.17: In the heptagon graph on vertices $\{a, b, c, d, e, f, g\}$: $\{a, g, f\}$ is not a constituency, because there is no subset of $\{b, c, d, e\}$ dominating these vertices; $\{g, f, b, c\}$ is not a constituency, because even though $\{a, d, e\}$ dominate the vertices, these 3 are not independent – they share an edge $(d, e)$; $\{b, g, f\}$ is a constituency, because the independent set $\{a, c, e\}$ dominates it.

Boolean networks are graphs where each vertex starts `true` or `false` (or, ON or OFF), and vertices successively update themselves based on their neighbors' values. This is a simple yet powerful model of computation – for example, the famous Conway's Game of Life is a Boolean network on an infinite grid graph. In other words, despite their simplicity, Boolean networks are sufficiently powerful to simulate any system which can be simulated[8]. Boolean networks may be either *synchronous* (updates are performed at time $t$ based on the status of each vertex at time $t - 1$) or *asynchronous* (updates are performed by each vertex in turn). Our work is in the latter setting: we view the MIS algorithm as a sequential update of a Boolean network

[8]Optionally through Conway's Game of Life, which is itself Turing-complete.

according to a permutation of the vertex set. A sequence of vertices (a “word”) can then be thought of as prescribing the ordering of this sequential update, and if the word leads to a MIS from any starting configuration we say the word is a fixing word (because after those updates the network is “fixed” in place). At a high level, our contributions are:

- Showing that it is computationally hard to (a) decide whether it is possible to reach every MIS from some starting configuration and (b) recognize fixing words or fixing permutations (permutations are words in which every vertex appears exactly once).
- Posing (and answering) several questions about those graphs which admit a fixing permutation (*permissible* graphs). We identify the heptagon as the smallest permissible graph, and exhibit large classes of permissible and non-permissible graphs, as well as proving that deciding whether a graph is permissible is computationally hard.
- Extending our study to directed graphs, where we search for *kernels*, which are analogous to maximal independent sets in undirected graphs.

## 1.5b Delaying Trains

*The Leonards are not on the platform at Long Beach train station, because the Long Island Railroad is delayed. Again.*

Chapter 5b is about choosing delays in train networks. When there is a disruption, carefully-chosen delays may be beneficial for certain passengers, who would otherwise miss some connections. Given a temporal graph (representing all trains in the network *after* the disruption) and a set of passengers (each specifying a starting vertex, an ending vertex, and a desired arrival time), we ask whether it is possible to delay some of the edges of the temporal graph to enable each passenger’s timely arrival. We call this problem DelayBetter (DB), and study it along with two variants: in $\delta$-DelayBetter, each delay must be of at most $\delta$; in Path DB, passengers fully specify the vertices they should visit on their journey. Our results are:

- On the positive side, polynomial-time algorithms for Path DB, and for DB and $\delta$-DB on trees (graphs without any cycles). This is then extended into an fpt algorithm for both problems parameterized by a measure of tree-likeness (the Feedback Edge Number of the graph) together with the number of passengers.
- On the negative side, NP-completeness of DB and $\delta$-DB even when $\delta$ and the lifetime of the graph are bounded and the graph is planar (can be drawn in the plane without edges crossing).

## 1.5c Partial Domination

The Leonards are gathered around a CCTV camera at 3rd Avenue and 27th street.

*E: I wonder how many of these cameras you would need to see every street corner.*

*D* nodding*: Make yourself a master of perspective [3, Note 530].*

*C: It can't be that hard, Manhattan's streets are in a grid.*

*E: Perhaps, but now suppose you and I owned different street corners, and I wanted to have cameras (ingenious inventions, I should say) positioned on those which I own to survey all which I own.*

*D: Do you suppose that this is any easier for our knowledge of the full area in which your peculiar estate is embedded?*

Chapter 5c is about the complexity of dominating part of a graph. We formalize this question as Partial Domination: given a graph $H$, a subset $T$ of its vertices, and a number $k$, is there a set $D$ of at most $k$ vertices in $T$ that dominates all others in $T$? There is a subtle difference between this problem and Dominating Set with the graph induced by $T$ in $H$ as its input: we may glean some information from the structure of $H$ about which vertices to include in $D$.

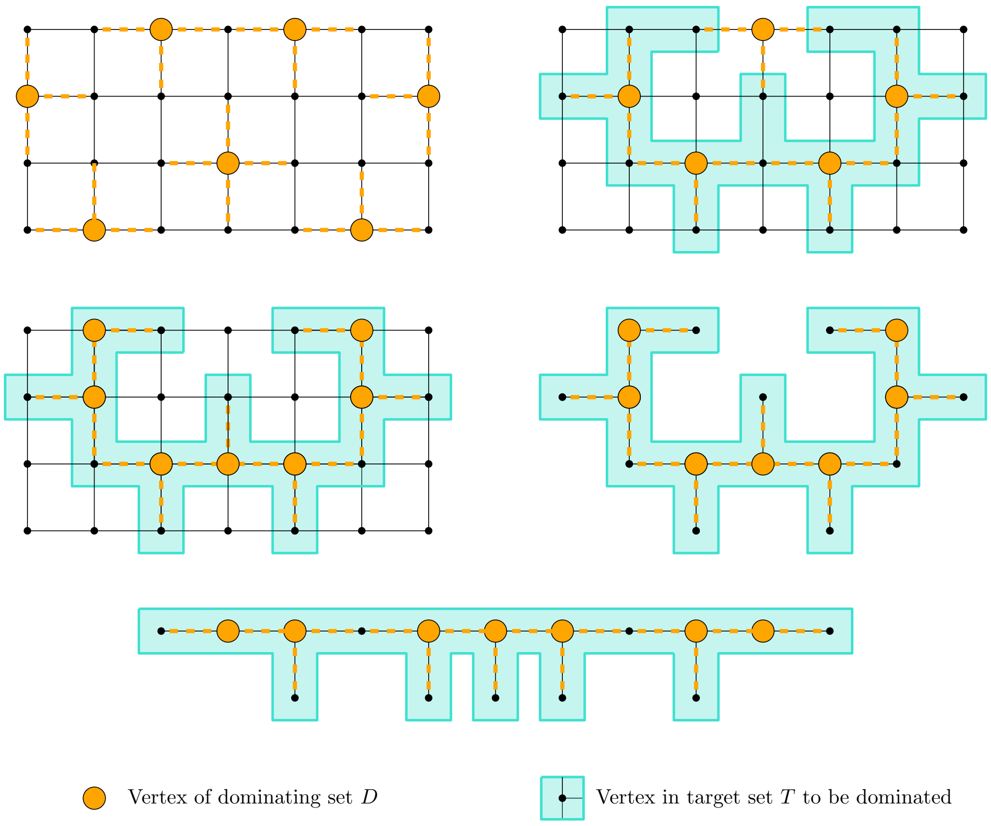

Vertex of dominating set $D$
Vertex in target set $T$ to be dominated

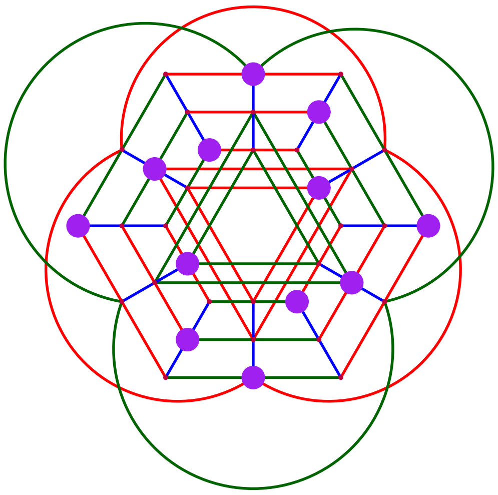
Figure 1.18: Top left: the host graph $H$ is a $4 \times 7$ grid, shown with a dominating set of size 7. Top right: the target set $T$ (shaded in turquoise) in $H$ can be dominated with just 5 vertices. Center left: dominating $T$ using only vertices from $T$ requires 7 vertices. Center right and bottom: the induced subgraph $H[T]$ requires the same number of vertices (7) to dominate, regardless of layout.

We study the complexity landscape of this problem when the "host graph" $H$ is restricted to some graph class $\mathcal{G}$ (for example, the class of all grid graphs). Partial

Domination restricted to $\mathcal{G}$ is at least as hard as Dominating Set is with the same restriction, and at most as hard as Dominating Set restricted to the class of induced subgraphs of members of $\mathcal{G}$ (its "hereditary closure"). The graph $H[T]$ shown at the bottom of Figure 1.18 is an induced subgraph of a grid, but is not itself a grid – so it belongs to the hereditary closure of the class of grids, and does not belong to the class of grids. For example, we construct a class $\mathcal{F}$ such that: Dominating Set is in P restricted to $\mathcal{F}$, Partial Domination is NP-intermediate restricted to $\mathcal{F}$, and Dominating Set is NP-complete restricted to the hereditary closure of $\mathcal{F}$.

## 1.5d A nifty problem: 1-in-3

At a gallery showing abstract art.



*D. : He made such beautiful figures [3, Note 1285]*

*E.* (gesturing at some pink dots)*: It is without a doubt the most beautiful solution to the problem [4, Letter XLII, p. 261]*

Chapter 5d is dedicated to a proof that a variant of Satisfiability, Triangle-free Tricolor Cubic Simple 1-in-3, is NP-complete. A rigorous definition of this problem variant is incompatible with a concise and informal summarization. A small illustration of 1-in-3 is shown in Figure 1.19.

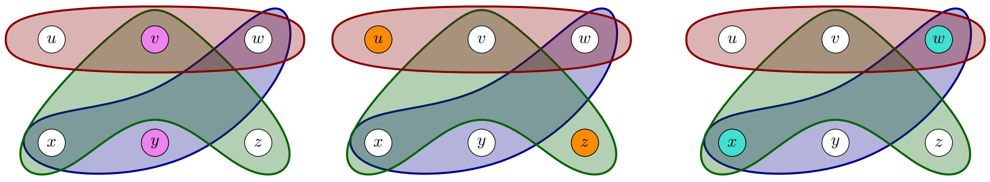


Figure 1.19: A 1-in-3 instance with 6 variables (or vertices) $X = \{u, v, w, x, y, z\}$ and 3 clauses (or hyperedges) $\mathcal{E} = \{(u, v, w), (x, y, w), (x, v, z)\}$. The objective is to select a set of vertices so that every clause (hyperedge) contains **exactly** one selected vertex. Left: $\{v, y\}$ is a valid solution. Center: $\{u, z\}$ is not a solution, since the blue curved constraint $\{x, y, w\}$ does not contain any of the chosen vertices. Right: $\{x, w\}$ is not a valid solution, since the blue curved constraint now contains **two** selected vertices.

## 1.6 Organization

The remainder of this thesis largely consists of published works or drafts for publication – in either case, written for a theoretical computer science audience. The curious reader who wished to get a flavor for the research herein without being exposed to technical detail has been warned: *hic sunt dracones.*

**Chapter 2** is based on [44], itself an extension of [45].

**Chapter 3** is based on [46], an extension of [47].

**Chapter 4** is based on [48], which extends [49].

**Chapter 5a** is an abridged version of [50].

**Chapter 5b** is based on [51].

**Chapter 5c** presents work which will hopefully be extended to a full paper in the future.

**Chapter 5d** is dedicated to proving the hardness of a specific combinatorial problem.

**Joint works** Chapter 3 is the result of collaborative work with Laura Larios-Jones, and will also appear in her doctoral thesis. Chapter 5a is the result of collaborative work with Maximilien Gadouleau. Chapter 5b is the result of collaborative work with Anouk Sommer. Each of these is prefaced with a fuller discussion of the author's own contributions to the work. All other chapters (including the present one) are solely attributed to the author and (different subsets of) his supervisory team.

# Chapter 2

# Payment scheduling in the Interval Debt Model

## 2.1 Introduction

A natural problem in the study of financial networks is that of whether and where a failure will occur if no preventative action is taken. We focus specifically on the flexibility that financial entities are afforded as regards the precise timing of their outgoings and for this purpose introduce the *Interval Debt Model (IDM)* in which a set of financial entities is interconnected by debts due within specific time intervals. In the IDM, a *payment schedule* specifies timings of payments to serve the debts. We examine the computational hardness of determining the existence of a schedule of payments with "good" properties, e.g., no or few bankruptcies, or minimizing the scale of remedial action. In particular, we establish how hardness depends on variations in the exact formalism of the model (to allow some small number of bankruptcies or insist on none at all) and on restrictions on the structure or lifetime of the input instance. A unique and novel feature of the IDM is its capacity to capture the temporal aspects of real-world financial systems; previous work has seldom explicitly dealt with this intrinsic facet of real-world debt.

**Financial Networks** Graph theory provides models for many problems of practical interest for analyzing (or administering) financial systems. For example, Eisenberg and Noe's work [11] abstracts a financial system to be a weighted digraph (in which each node is additionally labeled according to the corresponding entity's assets). The authors of that work are focused on the existence and computation of a *clearing vector*, which is essentially a set of payments among nodes of the graph which can be executed synchronously without violating some validity constraints. Their model provides the basis for much subsequent work in the network-based analysis of financial systems: it has been adapted to incorporate default costs [52], Credit Default Swaps [53] (CDSs) (that is, derivatives through which banks can bet on the default of another bank in the system) and the sequential behavior of bank defaulting in real-world financial networks [14].

An axiomatic aspect of Eisenberg and Noe's model is the so-called *principle of* *proportionality*: that a defaulting bank pays off each of its creditors proportionally to the amount it is owed. Some recent work has considered alternative payment schemes, which allow, for example, paying some debts in full and others not at all (so-called *non-proportional payments*). For example, Bertschinger, Hoefer and Schmand [13] study financial networks in a setting where each node is a rational agent which aims to maximize flow through itself by allocating its income to its debts. The focus of that work is on game-theoretic questions, such as the price of anarchy, or the existence, properties, and computability of equilibria. Papp and Wattenhoffer [54] also study a non-proportional setting, additionally incorporating CDSs.

Complementing the decentralized, game-theoretic approach is the question of the (centralized) computability of a *globally* "good" outcome through bailout allocation [12] (also called cash injection [55]), or timing default announcements [14], among other operations. In such works, the prototypical objective is to minimize the number of bankruptcies; related measures include total market value [12], or systemic liquidity [55]. Egressy and Wattenhoffer [12] focus solely on computational complexity of leveraging bailouts to optimize a range of objectives, in a setting which incorporates proportional payments and default costs. Kanellopoulos, Kyropoulou and Zhou [55, 56] apply both a game-theoretic and a classical complexity perspective to two mechanisms: debt forgiveness (deletion of edges in the financial network) and cash injection (bailouts). Notably, in this work the central authority may remove debts in a way which may be detrimental to certain individuals, but beneficial to the total systemic liquidity.

Previous research on financial networks has also drawn from ecology [57], statistical physics [58] and Boolean networks [59].

A central motivation of financial network analysis is to inform central banks' and regulators' policies. The concepts of *solvency* and *liquidity* are core to this task: a bank is said to be *solvent* if it has enough assets (including, e.g., debts owed to it) to meet all its obligations; and it is said to be *liquid* if it has enough liquid assets (that is, cash) to meet its obligations on time. An illiquid but solvent bank may exist even in modern interbank markets [60]. In such cases, a central bank may act as a *lender of last resort* and extend loans to such banks to prevent them defaulting on debts [61, 60]. The optimal allocation of bailouts to a system in order to minimize damage has also been studied as an extension of Eisenberg and Noe's model [62]. Here, bailouts refer to funds provided by a third party (such as a government) to entities to help them avoid bankruptcy.

**Temporal Graphs** Temporal graphs are graphs whose underlying connectivity structure changes over time. Such graphs allow us to model real-world networks which have inherent dynamic properties, such as transportation networks [63], contact networks in an epidemic [22, 47] and communication networks; for an overview see [64, 65]. Most commonly, following the formulation introduced by Kempe, Kleinberg and Kumar [66], a temporal graph has a fixed set of vertices together with edges that appear and disappear at integer times up to an (integer) lifetime. Often, a natural extension of a problem on static graphs to the temporal setting yields a computationally harder problem; for example, finding node-disjoint paths in a temporal graph remains NP-complete even when the underlying graph is itself a path [67], and finding a temporal vertex cover remains NP-complete even on star temporal graphs [68].

**Contributions** In this chapter we present a novel framework, the *Interval Debt Model* (IDM), for considering problems of bailout allocation and payment scheduling in financial networks by using temporal graphs to account for the isochronal aspect of debts between financial entities (previous work has almost exclusively focused on static financial networks). In particular, the IDM offers the flexibility that entities can pay debts earlier or later, within some agreed interval. We introduce several natural problems and problem variants in this model and show that the tractability of such problems depends greatly on the network topology and on the restrictions on payments (i.e., the admission or exclusion of partial and fractional payments on debts).

Our work explores the natural question of whether and how payments can be scheduled to avert large-scale failures in financial networks. Broadly, we establish that computing a zero-failure schedule (a *perfect schedule*) is NP-complete even when the network topology is highly restricted, unless we admit fractional payments, in which case determining the existence of a perfect schedule is tractable in general. Interestingly, if we allow a small number $k$ of bankruptcies to occur then every problem variant is computationally hard even on inputs with $O(1)$ nodes. This can be thought of analogously to Max 2SAT being strictly harder than 2SAT (unless P=NP) despite being a "relaxation": in Max 2SAT we allow up to $k$ clauses to not be satisfied. Furthermore, in the setting where we insist not only on payments being for integer amounts but more strongly that any payment is for the full amount of the corresponding debt, finding a perfect schedule is NP-complete even if there are only four nodes.

We begin by introducing, first by example and then formally, the Interval Debt Model in Section 2.2. In Section 2.3 we present our results: in Sections 2.3.1 to 2.3.3 we establish some sufficient criteria for NP-hardness for each of the problems we consider; and in Section 2.3.4 we present two polynomial-time algorithms. Our conclusions and directions for further research are given in Section 2.4.

## 2.2 The Interval Debt Model

In this section, we introduce (first by example and then formally) the *Interval Debt Model*, a framework in which temporal graphs are used to represent the collection of debts in a financial system.

### 2.2.1 An illustrative example

As an example, consider a tiny financial network consisting of the 3 banks $u$, $v$ and $w$ with €30, €20 and €10, respectively, in initial external assets and where there are the following inter-bank financial obligations:

- bank $u$ owes bank $v$ €20 which it must pay by time 3 and €15 which it must pay at time 4 or time 5 (note that all payments must be made at integer times)
- bank $v$ must pay bank $w$ a debt of €25 at time 2 exactly
- bank $w$ must pay €25 to bank $v$ between times 4 and 6 (that is, at time 4, 5 or 6).

A graphical representation of this system is shown in Fig. 2.1 (the descriptive notation used should be obvious and is retained throughout the chapter).

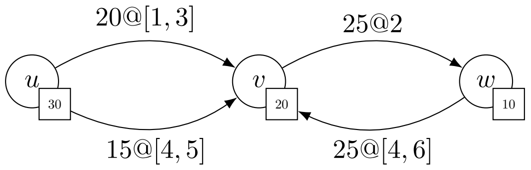


Figure 2.1: A simple instance of the Interval Debt Model (IDM). Numbers in square boxes represent the initial external assets of the node (for example, €30 for node $u$), directed edges represent debts, and the label on an edge represents the terms of the associated debt (for example, $u$ must pay $v$ €20 between time 1 and time 3).

Several points can be made about this system: node $u$ is *insolvent* as its €30 in initial external assets are insufficient to pay all its debts; node $v$ may be *illiquid* for it may default on part of its debt to $w$, e.g., if $u$ pays all of its first debt at time 3, or may remain liquid, e.g., if it receives at least €5 from $u$ by time 2; and node $w$ is solvent and certain to remain liquid in any case. The choices made by the various banks, in the form of a (payment) schedule, clearly affect the status of the overall financial system. Note that solvency is determined solely by whether sufficient funds exist whereas liquidity depends upon when debts are paid and owed.

One may ask several questions about our toy financial system such as: Are partial payments allowed (e.g., $u$ paying €18 of the €20 debt at time 1, and the rest later)? If so, are non-integer payments allowed? Can money received be immediately forwarded (e.g., $u$ paying $v$ €20 at time 2 and $v$ paying $w$ €25 at time 2)? Does $v$ necessarily have to pay its debt to $w$ at time 2 if it has the liquid assets to do so? We now expand upon these questions and specify in detail the setting we consider in the remainder of the chapter. Note that throughout the chapter we use the euro € as our monetary unit of resource even though, as we will see, we have a variant of the Interval Debt Model within which payments can be made for any rational fraction of a euro. We often prefix monetary payments with the symbol € to make our proofs more readable.

### 2.2.2 Formal setting

Formally, an *Interval Debt Model* (*IDM*) instance is a 3-tuple $(G, D, A^0)$ as follows.

- $G = (V, E)$ is a finite digraph with the set of $n$ nodes (or, alternatively, *banks*) $V = \{v_i : i = 1, 2, \ldots, n\}$ and the set of $m$ directed labelled edges $E \subseteq V \times V \times \mathbb{N}$, with the edge $(u, v, id) \in E$ denoting that there is an edge, or *debt*, whose label is $id$, from the *debtor* $u$ to the *creditor* $v$. We can have multi-edges but the labels of the edges from some node $u$ to some node $v$ must be distinct and form a contiguous integer sequence $0, 1, 2, \ldots$. We refer to the subset of edges directed out of or in to some specific node $v$ by $E_{\text{out}}(v)$ and $E_{\text{in}}(v)$, respectively. We also refer to the simple undirected graph obtained from $G$ by ignoring the multiplicity of and orientations on directed edges as the *footprint* of $G$.
- $D : E \to \{(a, t_1, t_2) : a, t_1, t_2 \in \mathbb{N} \setminus \{0\}, t_1 \leq t_2\}$ is the *debt function* which associates *terms* to every debt (ordinarily, we abbreviate $D((u, v, id))$ as $D(u, v, id)$). Here, if $e$ is a debt with terms $D(e) = (a, t_1, t_2)$ then $a$ is the *monetary amount* (or *monetary debt*) to be paid and $t_1$ (resp. $t_2$) is the first (resp. last) time at which (any portion of) this amount can be paid. For any debt $e \in E$, we also write $D(e) = (D_a(e), D_{t_1}(e), D_{t_2}(e))$. For simplicity of notation, we sometimes denote the terms $D(e) = (a, t_1, t_2)$ by $a@[t_1, t_2]$ or by $a@t_1$ when $t_1 = t_2$ (as we did in Fig. 2.1); also, for simplicity, we sometimes just refer to $a@[t_1, t_2]$ as the debt.
- $A^0 = (c^0_{v_1}, c^0_{v_2}, \ldots c^0_{v_n}) \in \mathbb{N}^n$ is a tuple with $c^0_{v_i}$ denoting the *initial external assets* (i.e. starting cash) of bank $v_i$.

We refer to the greatest time-stamp $T$ that appears in any debt for a given instance as the *lifetime* and assume that all network activity ceases after time $T$. The instance shown in Fig. 2.1, which has lifetime $T = 6$, is formally given by: $V = \{u, v, w\}$, $E = \{(u, v, 0), (u, v, 1), (v, w, 0), (w, v, 0)\}$, $D(u, v, 0) = (20, 1, 3)$, $D(u, v, 1) = (15, 4, 5)$, $D(v, w, 0) = (25, 2, 2)$, $D(w, v, 0) = (25, 4, 6)$ and $A^0 = (c^0_u, c^0_v, c^0_w)$, where $c^0_u = 30$, $c^0_v = 20$ and $c^0_w = 10$. Similarly, the instance shown in Fig. 2.2 has lifetime $T = 2$ and is given by $V = \{u, v, w\}$, $E = \{(u, v, 0), (v, w, 0)\}$, $D(u, v, 0) = (1, 1, 2)$, $D(v, w, 0) = (1, 1, 1)$ and $A^0 = (c^0_u, c^0_v, c^0_w)$, where $c^0_u = 1$, $c^0_v = 0$ and $c^0_w = 0$.

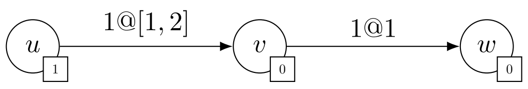


Figure 2.2: An IDM instance for which every schedule is described by four payment values $p^1_{(u,v,0)}$, $p^1_{(v,w,0)}$, $p^2_{(u,v,0)}$ and $p^2_{(v,w,0)}$.

The *size* of the instance $(G, D, A^0)$ is defined as $n + m + \log(T) + \beta$, where $\beta$ is the maximum number of bits needed to encode any of the (integer) numeric values appearing as the monetary amounts in the debts. Note that in what follows, we usually do not mention the label $id$ of a debt $(u, v, id)$ but just refer to the debt as $(u, v)$ when this causes no confusion.

### 2.2.3 Schedules

Given an IDM instance $(G, D, A^0)$, a (*payment*) *schedule* $\sigma$ describes the times at which the banks transfer *assets* to one another via *payments*. Formally, a schedule $\sigma$

is a set of $|E|T$ *payment values* $p_e^t \geq 0$, one for each edge-time pair $(e, t)$ (note that no payments are made at time 0). Equivalently, a schedule can be expressed as an $|E| \times T$ matrix $S$ with the payment values $p_e^t$ the entries of the matrix. The value $p_e^t$ is the monetary amount of the debt $e$ paid at time $t$. Our intention is that at any time $1 \leq t \leq T$, every payment value $p_e^t > 0$ of a schedule $\sigma$ is paid by the debtor of $e$ to the creditor of $e$, not necessarily for the full monetary amount $D_a(e)$ but for the amount $p_e^t$. A schedule for the instance of Fig. 2.2 consists of the four payments values $p^1_{(u,v,0)}$, $p^1_{(v,w,0)}$, $p^2_{(u,v,0)}$ and $p^2_{(v,w,0)}$. Note that, using the above representation of a schedule $\sigma$, we might have a large number of zero payments. Therefore, for simplicity of presentation, in the remainder of the chapter we specify schedules by only detailing the non-zero payments. An example schedule for the IDM instance in Fig. 2.2 is then $p^1_{(u,v,0)} = 1$, $p^1_{(v,w,0)} = 1$.

We now introduce some auxiliary variables which are not strictly necessary but help us to concisely express constraints on and properties of schedules. For nodes $u, v \in V$ and time $0 \leq t \leq T$, the following values are with respect to some specific schedule.

- Denote by $I_v^t$ the total monetary amount of incoming payments of node $v$ at time $t$.
- Denote by $O_v^t$ the total monetary amount of outgoing payments (expenses) of node $v$ at time $t$.
- We write $p^t_{u,v}$ to denote the total amount of all payments made from debtor $u$ to creditor $v$ at time $t$ in reference to *all* debts from $u$ to $v$; that is, $p^t_{u,v} = \sum_i p^t_{(u,v,i)}$.
- The vector $A^0 = (c^0_{v_1}, c^0_{v_2}, \ldots, c^0_{v_n})$ specifies the initial external assets (cash) of each node at time 0. For $t > 0$, we denote by $c_v^t$ node $v$'s cash assets at time $t$; that is, $c_v^t = c_v^{t-1} + I_v^t - O_v^t$.

For clarity, we refer to the starting cash of banks as "initial external assets" and to liquid assets in general as cash assets. By cash assets 'at time $t$' (resp. 'prior to time $t$') we mean after all (resp. before any) of the payments associated with time $t$ have been executed. Cash assets at time 0 are then precisely the initial external assets at time 0 (possibly supplemented by some bailout, as we shall see later).

We have been a little vague so far as regards the form of the payment values in any schedule and have not specified whether these values are integral, rational or do not necessarily equal the full monetary amount of the debt. As we detail below, we have variants of the model covering different circumstances (with perhaps the standard version being when payment values are integral but do not necessarily equal the full monetary amount of the debt).

Recall the example schedule from Fig. 2.2, which we can represent as $p^1_{u,v} = 1$, $p^1_{v,w} = 1$. As we shall soon see, the payments in this schedule can be legitimately discharged in order to satisfy the terms of all debts but in general this need not be the case. However, there might be schedules that are not *valid*, as well as valid schedules in which banks default on debts (that is, go *bankrupt*). We deal with the key notions of validity and bankruptcy now.

**Definition 2.1.** A schedule is *valid* if it satisfies the following properties (for any debt $e$, terms $D(e) = (a, t_1, t_2)$ and node $v$):

- all payment values are non-negative; that is, $p_e^t \geq 0$, for $1 \leq t \leq T$
- all cash asset values (as derived from payment values and initial external assets) are non-negative; that is, $c_v^t \geq 0$, for $0 \leq t \leq T$
- no debts are overpaid; that is, $\sum_{t=1}^{T} p_e^t \leq a$
- no debts are paid too early; that is, $\sum_{t=1}^{t_1-1} p_e^t = 0$.

Given some IDM instance, some schedule and some debt $e$ with terms $D(e) = (a, t_1, t_2)$, the debt $e$ is said to be *payable* at any time in the interval $[t_1, t_2 - 1]$. At time $t_2$, $e$ is said to be *due*. At time $t_2 \leq t \leq T$, if the full amount $a$ has not yet been paid (including payments made at time $t_2$) then $e$ is said to be *overdue* at time $t$. A debt is *active* whenever it is payable, due or overdue. However, a bank is said to be *withholding* if, at some time $1 \leq t \leq T$, it has an overdue debt and sufficient cash assets to pay (part of, where fractional or partial payments are permitted; see below) the debt. If any bank is withholding (at any time) in the schedule then the schedule is not valid.

So, for example and with reference to the IDM instance in Fig. 2.2, if, according to some schedule, bank $u$ pays 1 to bank $v$ at time 1 but $v$ makes no payment to $w$ at time 1 then $v$ is withholding and the schedule is not valid.

**Definition 2.2.** With reference to some schedule, a bank is said to be *bankrupt* (at time $t$) if it is the debtor of an overdue debt (at time $t$). We say that a schedule has $k$ *bankruptcies* if $k$ distinct banks are bankrupt at some time in the schedule (the times at which these banks are bankrupt might vary). A bank may *recover* from bankruptcy if it subsequently receives sufficient income to pay off all its overdue debts.

**Definition 2.3.** A bank $v$ is said to be *insolvent* if all its assets (that is, the sum of all debts due to $v$ and of $v$'s initial external assets) are insufficient to cover all its obligations (that is, the sum of all debts $v$ owes). Formally, $v$ is insolvent if

$$c_v^0 + \sum_{e \in E_{\text{in}}(v)} D_a(e) \quad < \sum_{e \in E_{\text{out}}(v)} D_a(e).$$

A bank which is insolvent will necessarily be bankrupt in any schedule.

We will not be concerned with the precise timing of bankruptcy or the recovery or not of any bank in this chapter.

We now detail three variants of the model (alluded to earlier) in which different natural constraints are imposed on the payment values.

**Definition 2.4.** In what follows, $e$ is an arbitrary debt and $1 \leq t \leq T$ some time.

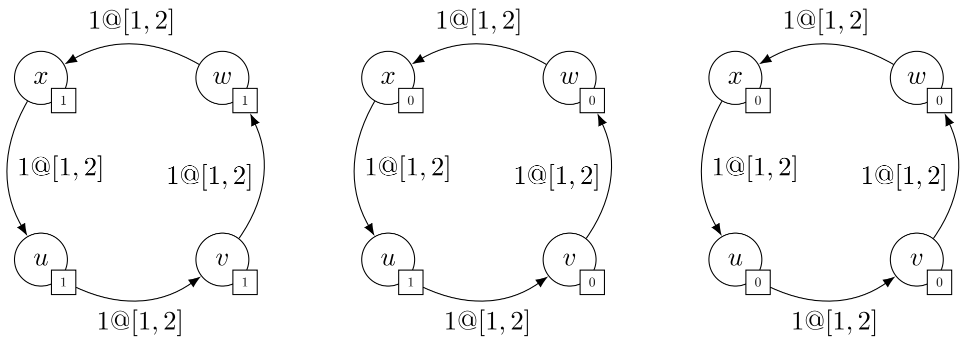
- In the *Fractional Payments* (*FP*) variant, the payment values may take rational values; that is, $p_e^t \in \mathbb{Q}$ and we allow payments for a smaller amount than the full monetary amount of $e$.
- In the *Partial Payments* (*PP*) variant, the payment values may take only integer values; that is, $p_e^t \in \mathbb{N}$ and we allow payments for a smaller amount than the full monetary amount of $e$.
- In the *All-or-Nothing* (*AoN*) variant, every payment value must fully cover the relevant monetary amount of $e$; that is, every payment value must be for the full monetary amount of $e$ or zero. So, $p_e^t \in \{D_a(e), 0\}$.

For example, the instance of Fig. 2.2 has the following valid schedules:

- (in all variants) the schedule above in which $p_{u,v}^1 = p_{v,w}^1 =$ €1 (all debts are paid in full at time 1)
- (in all variants) the schedule in which $p_{u,v}^2 = p_{v,w}^2 =$ €1 (all debts are paid in full at time 2)
  - under this schedule, node $v$ is bankrupt at time 1 as €1 of the debt $(v, w, 0)$ is unpaid and that debt is overdue
- (in the FP variant only) for every $a \in \mathbb{Q}$, where $0 < a < 1$, the schedule in which $p_{u,v}^1 = p_{v,w}^1 =$ €$a$ and $p_{u,v}^2 = p_{v,w}^2 =$ €$1 - a$
  - under each of these schedules, node $v$ is bankrupt at time 1 as €$1 - a$ of the debt $(v, w, 0)$ is unpaid and that debt is overdue.

It is worthwhile clarifying the concepts of instant forwarding and payment-cycles. We emphasize that we allow a bank to instantly spend income received. Note that in any valid schedule for the instance in Fig. 2.2, $v$ *instantly forwards* money received from $u$ to $w$ (so as not to be withholding); so, the cash assets of $v$ never exceed 0 in any valid schedule. This behaviour is consistent with the Eisenberg and Noe model [11] in which financial entities operate under a single clearing authority which synchronously executes payments. Indeed, in such cases a payment-chain of any length is permitted and the payment takes place instantaneously regardless of chain length.

Furthermore, and still consistent with the Eisenberg and Noe model, there is the possibility of a *payment-cycle* which is a set of banks $\{u_1, u_2, \ldots, u_c\}$, for some $c \geq 2$, with a set of debts $\{e_i = (u_i, u_{i+1}, l_i) : 1 \leq i \leq c - 1\} \cup \{e_c = (u_c, u_0, l_c)\}$ so that at some time $t$, all debts are active yet none has been fully paid and where each bank makes a payment, at time $t$, of the same value $a$ towards its debt. As an illustration, Fig. 2.3 shows three 'cyclic' IDM instances, all with lifetime $T = 2$. By our definition of a valid schedule, the schedule $p_{u,v}^1 = p_{v,w}^1 = p_{w,x}^1 = p_{x,u}^1 =$ €1, forming a payment-cycle, is valid in all three instances. This is intuitive for Fig. 2.3a, where each node has sufficient initial external assets available to pay all its debts in full at any time, irrespective of income. In Fig. 2.3b, we may imagine that the €1 moves from node $u$ along the cycle, satisfying every debt at time 1. This is a useful abstraction but not strictly accurate: rather, we should imagine that all four banks simultaneously order payments forward under a single clearing system. The clearing system calculates the balances that each bank would have with those payments executed, ensures they are all non-negative (one of our criteria for schedule validity) and then executes the payments by updating all accounts simultaneously. This distinction is significant when we consider Fig. 2.3c in which no node has any initial external assets. A clearing system ordered to simultaneously pay all debts would have no problem doing so in the Eisenberg and Noe model and in our model this constitutes a valid schedule. We highlight that there also exist valid schedules for the instance in Fig. 2.3c in which all four banks go bankrupt, one schedule being where all payments at any time are 0: here, no bank is withholding (they all have zero cash assets), so the schedule is valid, but every bank has an overdue debt and so is bankrupt.



(a) All nodes start with €1. (b) Only $u$ starts with €1. (c) All nodes start with €0.

Figure 2.3: Examples illustrating the behaviour of cycles in the IDM. In all instances shown the schedule in which all nodes pay their debts in full at time 1 is valid.

We use payment-cycles throughout our constructions in a context such as that in Fig. 2.4. Here, a valid schedule is where all nodes pay their corresponding debts in full at time $t$. The effect is that the €1 of cash assets at node $u$ is 'transferred' to €1 of cash assets at node $v$.

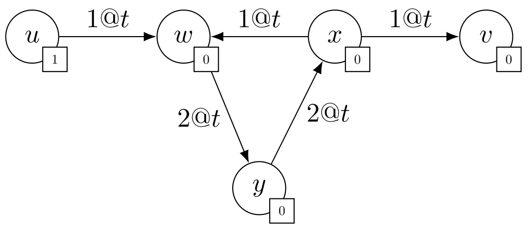


Figure 2.4: Using a payment-cycle to effectively transfer €1 of assets from node $u$ to node $v$.

### 2.2.4 Canonical instances

We wish to replace certain IDM instances with equivalent yet simpler ones. For example, consider the instance given in Fig. 2.1 but where every time-stamp in the instance is multiplied by a factor of 100 (so that, for example, the debt from $w$ to $v$ becomes $25@[400, 600]$). This 'inflated' instance is in essence equivalent to the original one but has a lifetime of 600.

**Definition 2.5.** Let $(G, D, A^0)$ be an instance. Then the set of time-stamps $\{t : D_{t_1}(e) = t \text{ or } D_{t_2}(e) = t, \text{ for some edge } e\}$ is the set of *extremal* time-stamps.

There is a simple preprocessing step such that we can assume that the lifetime $T$ of any IDM instance is polynomially bounded in $n$ and $m$ (that is, the numbers of banks and debts, respectively). This preprocessing step modifies the instance such that every $1 \leq t \leq T$ is an extremal time-stamp with the process being to simply omit non-extremal time-stamps and then compact the remaining time-stamps. Observe that this procedure is such that any valid schedule in the original IDM instance can be transformed into a valid schedule in the compacted instance so that these schedules have the same number of bankruptcies, eventual assets and so forth, and vice versa when the compacted instance is expanded into the original instance. Hence, we need not consider pathological cases in which the lifetime is, say, exponential in the number of nodes and debts. Given this restriction, we can now revise the notion of the size of an IDM instance to say that it is $n + m + \beta$ where $n$ is the number of banks, $m$ is the number of debts and $\beta$ is the maximum number of bits needed to encode any of the numeric values appearing as monetary amounts of debts.

**Lemma 2.6.** For any given IDM instance and any schedule, in any of the FP, PP or AoN variants, it is possible in polynomial-time both to check whether the schedule is valid and to compute the number of bankruptcies under the schedule.

*Proof sketch.* It is possible to iterate over the schedule once and calculate: the cash assets of every node, and which debts are overdue at each time-stamp. Computing the set $\{v | v \text{ has some overdue debt under } \sigma\}$ is then straightforward, and the number of bankruptcies is the cardinality of that set.

It remains to check the validity of the schedule. We can efficiently verify that there are:

**No negative assets:** verify that for any $u$ and $t$, $c_u^t \geq 0$.

**No withholding banks:** iterate once over the debts overdue at each time. If the debt $e = (u, v, i)$ is overdue at time $t$, verify that $c_u^t$ is insufficient to make a payment toward $e$ (i.e. $c_u^t = 0$ in the FP or PP model, or $c_u^t < D_a(e)$ in the AoN model).

**No overpaid debts:** iterate over all debts and ensure payments made with reference to each are no more than the debt amount.

**No debts paid early:** ensure $p_e^t = 0$ for any $t < D_{t_1}(e)$, for each debt $e$.

□

### 2.2.5 Problem definitions

We now define some decision problems with natural real-world applications.

Bankruptcy Minimization

Instance: an IDM instance $(G, D, A^0)$ and an integer $k$

Yes-instance: an instance for which there exists a valid schedule $\sigma$ such that at most $k$ banks go bankrupt at some time in the schedule $\sigma$.

Perfect Scheduling

Instance: an IDM instance $(G, D, A^0)$

Yes-instance: an instance for which there exists a valid schedule $\sigma$ such that no debt is ever overdue in $\sigma$; that is, a *perfect schedule*.

Bailout Minimization

Instance: an IDM instance $(G, D, A^0)$ (with $n$ banks) and an integer $b$

Yes-instance: an instance for which there exists a positive bailout vector $B = (b_1, b_2, \ldots, b_n)$ with $\sum_{i=1}^{n} b_i \leq b$ and valid schedule $\sigma$ such that $\sigma$ is a perfect schedule for the instance $(G, D, A^0 + B)$.

The problem Perfect Scheduling is equivalent to the Bailout Minimization problem where $b = 0$ and to the Bankruptcy Minimization problem where $k = 0$.

Bankruptcy Maximization

Instance: an IDM instance $(G, D, A^0)$ and an integer $k$

Yes-instance: an instance for which there exists a valid schedule $\sigma$ such that at least $k$ banks go bankrupt at some time in the schedule $\sigma$.

The problem Bankruptcy Maximization is interesting to consider for quantifying a 'worst-case' schedule where banks' behavior is unconstrained beyond the terms of their debts.

All of the problems above exist in the AoN, PP and FP variants and are in NP: for every yes-instance, there exists a witness schedule, polynomial in the size of the input, the validity of which can be verified in polynomial time (see Lemma 2.6).

Every valid PP schedule is a valid FP schedule whereas not every valid AoN schedule is a valid PP schedule. In an AoN schedule, a bank may go bankrupt while still having assets (insufficient to pay off any of its debts) whereas this is prohibited in any PP schedule as that bank would be withholding. If we restrict the instances to only those in which for every debt $e$, $D_a(e) = 1$ then every valid AoN schedule for that instance is a valid PP schedule and a valid FP schedule.

We call a digraph $G$ from some IDM instance a *multiditree* whenever the footprint of $G$ is a tree. We call a multiditree in which every edge is directed away from the root a *rooted out-tree* (or just *out-tree*). By an *out-path* we mean an out-tree where the footprint is a path and the root is either of the endpoints. We take this opportunity to note that an out-tree is both a directed acyclic graph (DAG) and a multiditree, but that not every multiditree DAG is an out-tree.

A summary of some of our upcoming results is given in Table 2.1 (with NP-c denoting 'NP-complete' and P denoting 'polynomial-time'). However, note that there are other, more nuanced results in what follows that do not feature in Table 2.1. Also, even though hardness for out-trees entails hardness for multiditrees and DAGs, we reference a separate result for the latter two settings where that is proven under different (stronger) constraints for the more general graph class. For example, our proof that AoN Bankruptcy Minimization is NP-complete on out-trees uses a construction requiring $T \geq 2$, but our proof of Theorem 2.7 has $T = 1$.

| problem | out-tree | multiditree | DAG | general case |
|---|---|---|---|---|
| FP Bankruptcy Minimization | ? | ? | NP-c (Thm 2.7) | NP-c (Thm 2.7) |
| PP Bankruptcy Minimization | ? | NP-c (Thm 2.10) | NP-c (Thm 2.7) | NP-c (Thm 2.7) |
| AoN Bankruptcy Minimization | NP-c (Thm 2.12) | NP-c (Thm 2.12) | NP-c (Thm 2.7) | NP-c (Thm 2.7) |
| FP Perfect Scheduling | P (Thm 2.15) | P (Thm 2.15) | P (Thm 2.15) | P (Thm 2.15) |
| PP Perfect Scheduling | P (Thm 2.17) | NP-c (Thm 2.10) | NP-c (Thm 2.9) | NP-c (Thm 2.9) |
| AoN Perfect Scheduling | NP-c (Thm 2.12) | NP-c (Thm 2.12) | NP-c (Thm 2.9) | NP-c (Thm 2.9) |
| FP Bailout Minimization | P (Thm 2.15) | P (Thm 2.15) | P (Thm 2.15) | P (Thm 2.15) |
| PP Bailout Minimization | P (Thm 2.17) | NP-c (Thm 2.10) | NP-c (Thm 2.9) | NP-c (Thm 2.9) |
| AoN Bailout Minimization | NP-c (Thm 2.12) | NP-c (Thm 2.12) | NP-c (Thm 2.9) | NP-c (Thm 2.9) |
| FP Bankruptcy Maximization | ? | ? | NP-c (Thm 2.13) | NP-c (Thm 2.13) |
| PP Bankruptcy Maximization | ? | ? | NP-c (Thm 2.13) | NP-c (Thm 2.13) |
| AoN Bankruptcy Maximization | NP-c (Thm 2.14) | NP-c (Thm 2.14) | NP-c (Thm 2.14) | NP-c (Thm 2.14) |

Table 2.1: Summary of results.

### 2.2.6 Discussion of the model

We describe here some notable differences (and similarities) of the IDM as compared with other studied models.

First and foremost, the IDM is a temporal model; the eponymous "interval debts" are its principal distinguishing feature when contrasted with other financial network models. The *timing* of payments, not their allocation to one payee or another, is the principal question. In fact, in Perfect Scheduling this is the *only* question.

As we shall see, under the restriction $D_{t_1} = D_{t_2}$ that problem (and its superproblem Bailout Minimization) become straightforwardly solvable in all variants. All of our hardness results arise from the expressivity of that degree of freedom (the scheduling of payments sooner or later). Indeed, in Bankruptcy Minimization and Bankruptcy Maximization that freedom remains, and the problems remain NP-complete under the same restriction $D_{t_1} = D_{t_2}$. Consequently, the results of other works which do not have a temporal component [56, 12, 54] do not straightforwardly carry over to the IDM.

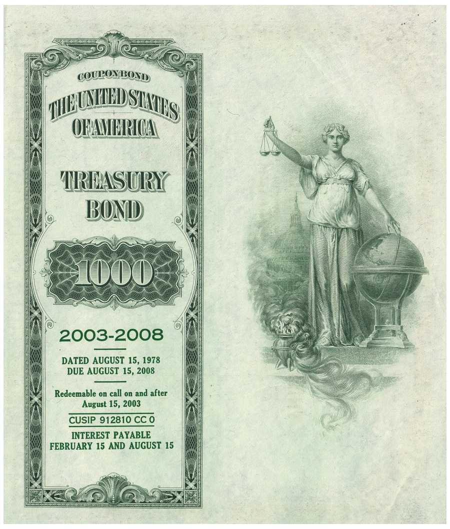

COUPON BOND
THE UNITED STATES OF AMERICA
TREASURY BOND
1000
2003-2008
DATED AUGUST 15, 1978
DUE AUGUST 15, 2008
Redeemable on call on and after August 15, 2003
CUSIP 912810 CC 0
INTEREST PAYABLE FEBRUARY 15 AND AUGUST 15

Figure 2.5: A real-life interval debt: this 1978 US government bond is payable between 2003 and 2008.

We take this opportunity to emphasize that interval debts are practically motivated; in particular, some real-world debts may be paid neither early nor late (see, e.g., Figure 2.5).

Non-proportional payments on debts are likewise nothing new to financial networks. Recent work has considered frameworks wherein *priorities* are associated with each debt, with higher-priority debts paid off before lower-priority debts [56, 54]. In such a setting, the priority of some debt may be either chosen by a regulatory authority or left to the individual agents. In the former case, the hardness of computing a solution which maximizes utility is of particular interest, whereas game-theoretic approaches are more relevant in the latter. Our focus in the present work is solely on questions of computational complexity from a centralized perspective.

We note that Bailout Minimization and its subproblem Perfect Scheduling remain unchanged as decision problems if bankruptcy in the IDM is redefined to require proportional payments, or immediate deletion of the bankrupt node. Both problems fundamentally ask whether a perfect schedule exists; consequently, the manner in which bankruptcy and defaulting are modeled in the IDM are irrelevant. If there is a perfect schedule $\sigma$, then under $\sigma$ all debts are by definition paid on time, in full (and hence proportionally). Conversely, if no such $\sigma$ exists, then a bankruptcy (however it is modeled) must occur, and we have a no-instance of the respective problems.

Lastly, we would like to comment briefly on the respective practical value of the AoN, PP and FP variants. The FP variant is quite intuitive for theoreticians, and yields our main tractable case. On the other hand, the PP variant realizes the practical constraint that arbitrarily small transfers are impractical, and may be of interest where a fungible but indivisible resource needs to be exchanged. Personal communication [69] suggests that, perhaps unexpectedly, the AoN variant may well be the one of most interest to the finance community. Unlike most other models, the AoN model has the unintuitive property that a bankrupt bank may retain some assets. We note that this modelling of bankruptcy is not required for any of our hardness of tractability proofs in the AoN model.

## 2.3 Our results

In this section we investigate the complexity of the problems presented above. We present our hardness results for Bankruptcy Minimization, Perfect Scheduling and Bankruptcy Maximization in Sections 2.3.1, 2.3.2 and 2.3.3, respectively, and then in Section 2.3.4 show that under certain constraints the problem Bailout Minimization and its subproblem Perfect Scheduling become tractable.

### 2.3.1 Hardness results for Bankruptcy Minimization

We begin with our core hardness result.

**Theorem 2.7.** For each of the AoN, PP and FP variants, the problem Bankruptcy Minimization is NP-complete, even when we restrict to IDM instances $(G, D, A^0)$ for which: $T = 1$; $G$ is a directed acyclic graph with a longest path of length 4, has out-degree at most 2 and has in-degree at most 3; the monetary amount of any debt is at most €3; and initial external assets are at most €3 per bank.

*Proof.* We build a polynomial-time reduction from the problem 3-Sat-3 to Bankruptcy Minimization so that all target instances satisfy the constraints in the statement of the theorem (the problem 3-Sat-3, defined below, was shown to be NP-complete in [70]).

3-SAT-3

Instance: a c.n.f. formula $\phi$ over $n$ Boolean variables $v_1, v_2, ..., v_n$ so that each of the $m$ clauses $c_1, c_2, ..., c_m$ has size at most 3 and where there are exactly 3 occurrences of $v_i$ or $\neg v_i$ in the clauses

Yes-instance: there exists a satisfying truth assignment for $\phi$.

We may (and do, throughout the chapter) restrict ourselves to those instances in which every literal appears at least once and at most twice (that is, both $v_i$ and $\neg v_i$ appear in some clause, for $1 \leq i \leq n$) and where no clause contains both a literal and its negation. We define the size of an instance $\phi$ to be $n$.

Suppose that we are given a 3-Sat-3 instance $\phi$ of size $n$. We construct an IDM instance $(G, D, A^0)$ as follows. For any variable $v_i$, denote by $count_{v_i}$ (resp. $count_{\neg v_i}$) the number of occurrences of the literal $v_i$ (resp. $\neg v_i$) in $\phi$ (of course, $count_{v_i} + count_{\neg v_i} = 3$). We build a digraph $G$ with:

- a *source node* $s_i$, for each variable $v_i$, so that this node has initial external assets €3 (every other type of node will have initial external assets €0)
- two *literal nodes* $x_i$ and $\neg x_i$, for each variable $v_i$
- a *clause node* $q_j$, for each clause $c_j$
- a *sink node* $d$.

We then add edges and debts as follows. For every $1 \leq i \leq n$:

- we add the debt $(s_i, x_i)$ with terms 3@1
- we add the debt $(s_i, \neg x_i)$ with terms 3@1
- we add the debt $(x_i, d)$ with terms $\mathit{count}_{\neg v_i}$@1 (note that the monetary amount to be paid is either €1 or €2)
- we add the debt $(\neg x_i, d)$ with terms $\mathit{count}_{v_i}$@1 (note that the monetary amount to be paid is either €1 or €2)
- for every $1 \leq j \leq m$:
  - we add the debt $(q_j, d)$ with terms 1@1
  - if the literal $v_i \in c_j$ then we add the debt $(x_i, q_j)$ with terms 1@1
  - if the literal $\neg v_i \in c_j$ then we add the debt $(\neg x_i, q_j)$ with terms 1@1.

Fig. 2.6 shows a sketch of this construction (where nodes without any depicted initial external assets start with €0). The IDM instance $(G, D, A^0)$ can clearly be built from $\phi$ in polynomial-time.

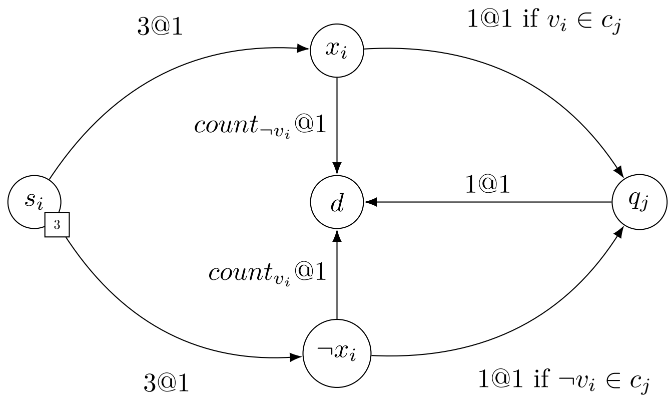


Figure 2.6: Construction sketch for an IDM instance from a given formula $\phi$. Note that each of $x_i$ and $\neg x_i$ owes €3 in total.

We claim that the instance $((G, D, A^0), 2n)$ of Bankruptcy Minimization as constructed above is a yes-instance of Bankruptcy Minimization (no matter which of the AoN, PP and FP variants we work with) iff $\phi$ is a yes-instance of 3-Sat-3.

Before we proceed, we have the following remark. Recall that for each $1 \leq i \leq n$, $\mathit{count}_{v_i} + \mathit{count}_{\neg v_i} = 3$; consequently, $\mathit{count}_{v_i} = 2$ iff $\mathit{count}_{\neg v_i} = 1$, and vice versa. For each $1 \leq i \leq n$, node $x_i$ (resp. $\neg x_i$) has a total monetary debt to the clause nodes of $\mathit{count}_{v_i}$ (resp. $\mathit{count}_{\neg v_i}$) and a total monetary debt to the sink node $d$ of $\mathit{count}_{\neg v_i}$ (resp. $\mathit{count}_{v_i}$); so, each literal node has a total monetary debt of €3.

**Claim 2.7.1.** If $\phi$ is a yes-instance of 3-SAT-3 then $((G, D, A^0), 2n)$ is a yes-instance of BANKRUPTCY MINIMIZATION.

*Proof.* Suppose that $\phi$ is satisfiable via some truth assignment $X$. Consider the schedule $\sigma$ for $(G, D, A^0)$ in which:

- every source node $s_i$ pays €3 (at time 1, as are all payments) to the literal node $x_i$ (resp. $\neg x_i$) if $X(v_i) = True$ (resp. $X(v_i) = False$)
- every literal node $x_i$ (resp. $\neg x_i$) for which $X(v_i) = True$ (resp. $X(v_i) = False$) pays all its debts in full
  - as remarked above, this literal node has total monetary debt €3 but, from above, it receives €3 from $s_i$
- every clause node pays its €1 debt to the sink node $d$
  - this is necessarily possible because $X$ is a satisfying truth assignment, meaning every clause node receives at least €1 from some literal node corresponding to a literal in that clause set to $True$ by $X$.

Note that $\sigma$ is valid in all three IDM variants. The total number of bankruptcies in $\sigma$ is $2n$: exactly $n$ bankrupt source nodes and exactly $n$ bankrupt literal nodes. Hence, if $\phi$ is satisfiable then the schedule $\sigma$ for $(G, D, A^0)$ results in at most (in fact, exactly) $2n$ bankruptcies. □

**Claim 2.7.2.** If $(G, D, A^0), 2n)$ is a yes-instance of BANKRUPTCY MINIMIZATION then $\phi$ is a yes-instance of 3-SAT-3.

*Proof.* Suppose that we have a schedule $\sigma$ for $(G, D, A^0)$ with at most $2n$ bankruptcies. Consider the set of all literals $L = \{v_1, \neg v_1, v_2, \neg v_2, ..., v_n, \neg v_n\}$ w.r.t. $\phi$. Define the set of *bankrupt literals* $B \subseteq L$ to consist of every literal whose corresponding (literal) node is bankrupt within $\sigma$. Define $X(w) = True$ iff $w \in L \setminus B$. We claim that $X$ is a (complete) truth assignment. Suppose it is not and that $X(v_i) = X(\neg v_i) = False$; so, both $x_i$ and $\neg x_i$ are bankrupt within $\sigma$. However, in any valid schedule (no matter what the IDM variant) every source node $s_i$ will necessarily go bankrupt and at least one of the literal nodes $x_i$ and $\neg x_i$ will go bankrupt. Thus, as we have at most $2n$ bankruptcies, our supposition is incorrect. Alternatively, suppose that $X(v_i) = X(\neg v_i) = True$; so, neither $x_i$ nor $\neg x_i$ is bankrupt within $\sigma$. But, as stated, this cannot be the case. So, $X$ is a truth assignment; moreover, $\sigma$ has exactly $2n$ bankruptcies with exactly one of any pair of 'oppositely-oriented' literal nodes bankrupt.

Suppose, for contradiction, that $X$ is not a satisfying assignment. So, there exists at least one clause, $c_j$ say, such that every literal in the clause is made $False$ by $X$. By definition of $X$, we have that every literal node corresponding to one of these literals is a bankrupt node. Any such literal node must receive €0 (as the 'oppositely-oriented' literal node is not bankrupt and receives €3); consequently, the clause node

$q_j$ receives €0 and is bankrupt. This yields a contradiction as we have exactly $2n$ bankrupt nodes (as detailed above). □

Consequently, $\phi$ is satisfiable iff the IDM instance $(G, D, A^0)$ admits a schedule with at most $2n$ bankruptcies (again, this holds for each IDM variant). This concludes our proof. □

Our next result is perhaps rather surprising in that we restrict to IDM instances $(G, D, A^0)$ where $G$ is a fixed digraph.

**Theorem 2.8.** For each of the AoN, PP and FP variants, the problem BANKRUPTCY MINIMIZATION is weakly NP-complete, even when we restrict to instances $((G, D, A^0), k)$ where $G$ is a fixed, specific digraph with 32 nodes and $k = 16$.

*Proof.* We build a polynomial-time reduction from EQUAL CARDINALITY PARTITION to BANKRUPTCY MINIMIZATION (EQUAL CARDINALITY PARTITION, defined below, was proven weakly NP-complete in [10]).

EQUAL CARDINALITY PARTITION

Instance: a multi-set of positive integers $S = \{a_1, a_2, ..., a_n\}$ where $n$ is even and with sum $sum(S) = 2k$

Yes-instance: there exist a partition of $S$ into two equal-sized sets $S_1$ and $S_2$ such that $sum(S_1) = sum(S_2) = k$.

The size of such an instance is $n\beta$ where $\beta$ is the least number of bits so that any integer $a_i$ can be represented in binary using $\beta$ bits.

Given an instance $S = \{a_1, a_2, ..., a_n\}$ of EQUAL CARDINALITY PARTITION (where $n \geq 1$), we construct the IDM instance $(G, D, A^0)$ that is illustrated in Fig. 2.7. Every appearance of the shaded node $p$ in Fig. 2.7 corresponds to the same single node, that we refer to as the sink, and thus this instance has 32 nodes in total. We use the symbol '$\infty$' to denote a suitably high monetary amount (though $2k + n + 1$ suffices) and $T = 10n + 7$. Our IDM instance can trivially be constructed from $S$ in time polynomial in the size of the instance $S$. We now show that $(G, D, A^0)$ admits a valid schedule with at most 16 bankruptcies iff $S$ is a yes-instance of EQUAL CARDINALITY PARTITION (no matter whether we are in the AoN, PP or FP variant). Until further stated, we will work solely within the PP variant and return to the AoN and FP variants later.

**Claim 2.8.1.** If the IDM instance $((G, D, A^0), 16)$ is a yes-instance of BANKRUPTCY MINIMIZATION then $S$ is a yes-instance of EQUAL CARDINALITY PARTITION.

*Proof.* Suppose that $(G, D, A_e^0)$ admits a valid schedule $\sigma$ with at most 16 bankruptcies. Note that the 14 nodes $m_1$, $m_3$, $m_4^A$, $m_6^A$, $m_8^A$, $m_{11}^A$, $m_{13}^A$, $m_{15}^A$, $m_4^B$, $m_6^B$, $m_8^B$, $m_{11}^B$, $m_{13}^B$ and $m_{15}^B$ are necessarily bankrupt in *every* valid schedule because they are

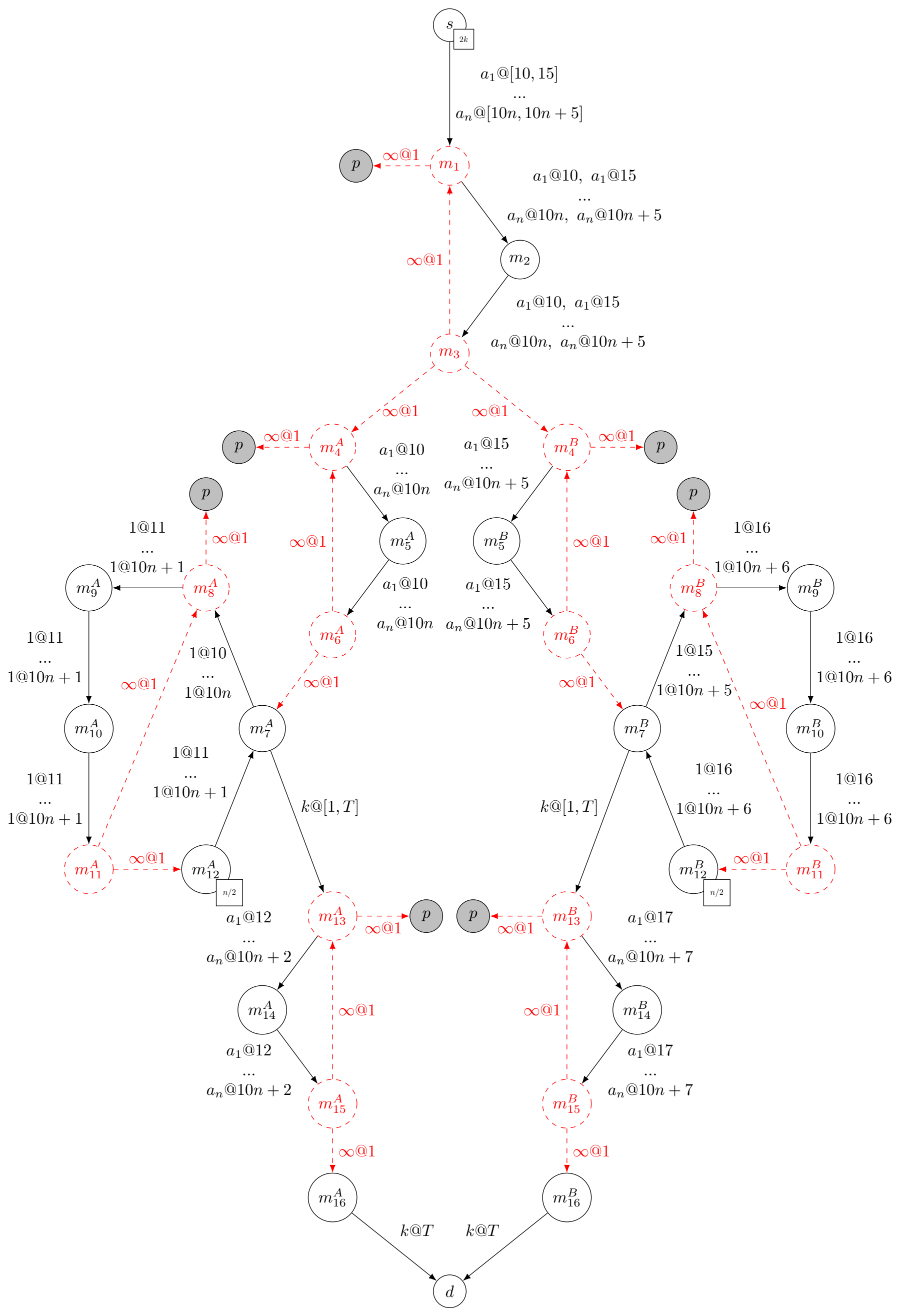


Figure 2.7: Construction of an IDM instance (with $k = 16$) corresponding to the EQUAL CARDINALITY PARTITION instance $S = \{a_1, \ldots, a_n\}$. Dashed red edges are "practically infinite" bankrupting debts.

debtors of debts of monetary amount $\infty$ at time 1 and no payments are made before time 10 (these nodes are dashed and highlighted in red in Fig. 2.7 as are the debts at time 1 of monetary amount $\infty$). Note also that these nodes can never have any cash assets at any time and nor can the nodes $m_2$, $m_5^A$, $m_9^A$, $m_{10}^A$, $m_{14}^A$, $m_5^B$, $m_9^B$, $m_{10}^B$ and $m_{14}^B$ (as they would otherwise be withholding). Consequently, we can only have at most another 2 nodes going bankrupt within $\sigma$. We begin by showing that one of $\{m_7^A, m_{12}^A\}$ must be bankrupt and one of $\{m_7^B, m_{12}^B\}$ must be bankrupt (which will account for all bankrupt nodes).

Suppose that none of the nodes $m_7^A$, $m_9^A$, $m_{10}^A$ and $m_{12}^A$ are bankrupt in $\sigma$. By considering $m_{12}^A$ at the times $t \in \{11, 21, \ldots, 10n+1\}$, on at least $\frac{n}{2}$ of these occasions $m_{12}^A$ must have received at least €1 via payments from $m_{11}^A$ (so as to service all its debts to $m_7^A$). Consider the first occasion $t' \in \{11, 21, \ldots, 10n+1\}$ that $m_{12}^A$ receives a non-zero payment from $m_{11}^A$ (note that all payments from $m_{11}^A$ to $m_{12}^A$ are made at some time from $\{11, 21, \ldots, 10n+1\}$). There must have been a payment of €1 from $m_{10}^A$ to $m_{11}^A$ at time $t'$ as well as payments of €1 from $m_9^A$ to $m_{10}^A$ and from $m_8^A$ to $m_9^A$ at time $t'$. So, $m_8^A$ must receive at least €1 from $m_7^A$ and $m_{11}^A$ at time $t'$. As $m_{11}^A$ makes a non-zero payment to $m_{12}^A$ at time $t'$, any payment from $m_{11}^A$ to $m_7^A$ at time $t'$ must be for some amount strictly less than €1. Consequently, $m_8^A$ must receive a non-zero payment from $m_7^A$ at time $t'$. The only way that this can happen is if there is an overdue debt from $m_7^A$ to $m_8^A$; that is, $m_7^A$ is bankrupt, which yields a contradiction. Hence, at least one of $m_7^A$, $m_9^A$, $m_{10}^A$ and $m_{12}^A$ is bankrupt. An analogous argument shows that at least one of $m_7^B$, $m_9^B$, $m_{11}^B$ and $m_{12}^B$ is bankrupt. Hence, exactly one of $m_7^A$, $m_9^A$, $m_{10}^A$ and $m_{12}^A$ is bankrupt and exactly one of $m_7^B$, $m_9^B$, $m_{11}^B$ and $m_{12}^B$ is bankrupt.

Suppose that $m_9^A$ is bankrupt. So, $m_{10}^A$ is necessarily bankrupt. Conversely, if $m_{10}^A$ is bankrupt then $m_9^A$ must be bankrupt also. Consequently, neither $m_9^A$ nor $m_{10}^A$ is bankrupt and we must have that either $m_7^A$ or $m_{12}^A$ is bankrupt and analogously either $m_7^B$ or $m_{12}^B$ is bankrupt. In particular, $s$, $m_2$, $m_5^A$, $m_{13}^A$, $m_{15}^A$, $m_5^B$, $m_{13}^B$ and $m_{15}^B$ are not bankrupt.

Let us turn to analysing the flow of resource via the schedule $\sigma$. As $\sigma$ is valid, we must have that both debts from $m_{16}^A$ and $m_{16}^B$ are paid on time with €$2k$ in total reaching $d$. The question is: does this resource consist of the €$2k$ emanating from $s$ or does it consist of resource emanating from $s$ but supplemented with resource emanating from $m_{12}^A$ or $m_{12}^B$? Let us look at the possible debt payments at time 10 (which is the earliest time that payments can be made). In particular, let us look at payments made by $m_6^A$ to $m_7^A$ at time 10. Note that all payments made from $m_6^A$ to $m_7^A$ are at a time from $\{10, 20, \ldots, 10n\}$.

Case $(a)$: An amount of €$x > 0$ is paid from $m_6^A$ to $m_7^A$ at time 10.

There are two essential sub-cases at time 10:

$(i)$ $m_7^A$ services its debt to $m_8^A$, which pays €1 to the sink, with perhaps €$y \geq 0$ paid from $m_7^A$ to $m_{13}^A$ and from there to the sink, so that €$x - y - 1 \geq 0$ resides at $m_7^A$ in cash assets at time 10

$(ii)$ $m_7^A$ does not service its debt to $m_8^A$ and pays €$x$ to $m_{13}^A$ with this payment immediately going to the sink, so that €0 resides at $m_7^A$ in cash assets at time 10; hence, $m_7^A$ is bankrupt (note that no cash assets can reside at $m_7^A$ at time 10 as otherwise $m_7^A$ would be withholding).

Now consider what happens at time 11. Suppose that we are in Case $(a.i)$. There must be a payment-cycle involving $m_8^A$, $m_9^A$, $m_{10}^A$ and $m_{11}^A$ and as $\frac{n}{2} \geq 1$, $m_{12}^A$ must service its debt to $m_7^A$, with perhaps $m_7^A$ paying €$z \geq 0$ to $m_{13}^A$ which is immediately paid to the sink. The €1 from $m_{12}^A$ does not supplement the resource emanating from $s$ but just 'replaces' €1 which was 'lost' to the sink at time 10. Note that the cash assets of $m_{12}^A$ at time 11 are $\frac{n}{2} - 1$.

Suppose that we are in Case (*a.ii*). Note that as $m_7^A$ is bankrupt, $m_{12}^A$ can never become bankrupt and so must service its debts when required. There are four possibilities as regards what happens at time 11 (bearing in mind the overdue debt from $m_7^A$ to $m_8^A$):

(1) $m_{12}^A$ pays €1 to $m_7^A$ which pays €1 to $m_8^A$ which immediately goes to the sink, with a payment-cycle involving $m_8^A$, $m_9^A$, $m_{10}^A$ and $m_{11}^A$

(2) $m_{12}^A$ pays €1 to $m_7^A$ which pays €1 to $m_8^A$ which pays €1 to $m_9^A$ which pays €1 to $m_{10}^A$ which pays €1 to $m_{11}^A$ which pays €1 to $m_8^A$ which immediately goes to the sink

(3) $m_{12}^A$ pays €1 to $m_7^A$ which pays €1 to $m_{13}^A$ which immediately goes to the sink, with a payment-cycle involving $m_8^A$, $m_9^A$, $m_{10}^A$ and $m_{11}^A$

(4) $m_{12}^A$ pays €1 to $m_7^A$ which pays €1 to $m_8^A$ which pays €1 to $m_9^A$ which pays €1 to $m_{10}^A$ which pays €1 to $m_{11}^A$ which pays €1 to $m_{12}^A$; that is, we have a payment cycle involving $m_7^A$, $m_8^A$, $m_9^A$, $m_{10}^A$, $m_{11}^A$ and $m_{12}^A$.

In (1-3) above, the €1 from $m_{12}^A$ does not supplement the resource emanating from $s$ but is lost to the sink (as are the €$x > 0$ at time 10). Note that in (3), the debt from $m_7^A$ to $m_8^A$ at time 10 is overdue and cannot be paid at time 11 as $m_7^A$ has no cash assets at time 10; so it remains overdue. In (4), again no supplement is made, although €$\frac{n}{2}$ still resides at $m_{12}^A$ in cash assets at time 11, and €$x > 0$ emanating from $s$ has been lost to the sink.

In both Case (*a.i*) and Case (*a.ii*), at time 12, the only possible non-payment-cycle payments involve $s$, $m_1$, $m_7^A$, $m_{13}^A$, $m_{14}^A$ and $m_{15}^A$ but any such payments do not affect whether the resources emanating from $m_{12}^A$ or $m_{12}^B$ supplement the resource emanating from $s$. In Case (*a.ii*.3), the debt from $m_7^A$ to $m_8^A$ at time 10 is overdue and so must be paid from the cash assets of $m_7^A$ at time 12, if it has any and unless all of these assets are paid to $m_{13}^A$. All such payments by $m_7^A$ will immediately go to the sink.

Case (*b*): No payment is made from $m_6^A$ to $m_7^A$ at time 10.

Consequently, $m_7^A$ cannot pay its debt to $m_8^A$ and becomes bankrupt. At time 11, $m_{12}^A$ must necessarily service its debt to $m_7^A$ and there are four possibilities with these possibilities being exactly the possibilities (1-4) in Case (*a.ii*). As before, at time 12, the only possible non-payment-cycle payments made involve $s$, $m_1$, $m_7^A$, $m_{13}^A$, $m_{14}^A$ and $m_{15}^A$ but any such payments do not affect whether the resources emanating from $m_{12}^A$ or $m_{12}^B$ supplement the resources emanating from $s$. Note that in (3), the debt from $m_7^A$ to $m_8^A$ at time 10 is still overdue at time 11 and cannot be paid at time 12 as $m_7^A$ has no cash assets at time 12; so it remains overdue. In (4), again no supplement is made, although €$\frac{n}{2}$ still resides at $m_{12}^A$ in cash assets at time 12.

So, the resources emanating from $m_{12}^A$ cannot supplement the resources emanating from $s$ at any time $t < 15$. An identical argument can be applied to time 15 so as to yield a similar conclusion as regards the resources emanating from $m_{12}^B$ at any time $t < 20$.

Consider the situation at time 20. With regard to the sub-network involving $m_6^A$, $m_7^A$, $m_8^A$, $m_9^A$, $m_{10}^A$, $m_{11}^A$, $m_{12}^A$ and $m_{13}^A$ (that is, the sub-network of study above), the situation is similar to that at time 10 except that there may be additional restrictions on what can happen given: a possible existing overdue debt from $m_7^A$ to $m_8^A$ (the debt due at time 10); possible non-zero cash assets at $m_7^A$; and possibly reduced cash assets at $m_{12}^A$ of $\frac{n}{2} - 1$. Note that if $m_7^A$ has cash assets prior to time 20 then this can be thought of as $m_7^A$ having acquired these assets from $m_6^A$ at time 20; that is, we are in Case $(a)$ above. Given the fact that the situation at time 20 is a restricted version of the situation at time 10 where the resources emanating from $m_{12}^A$ could not supplement those emanating from $s$, the same is true again. By analysing each time $t = 25, 30, 35, \ldots$, we can see that no resources emanating from either $m_{12}^A$ or $m_{12}^B$ can supplement those emanating from $s$. Hence, in order to secure total cash assets of €$2k$ at $d$ after time $T$, we need that all of the €$2k$ resource emanating from $s$ reaches $d$; that is, none of it is lost to the sink en route (although some of it might have been 'replaced' as per Case $(a.i)$).

We can now repeat the above analysis except that now we know that we cannot lose resource emanating from $s$ unless it is replaced as in Case $(a.i)$. This simplifies things considerably. If $m_7^A$ has €$x > 0$ at time $t \in \{10, 20, \ldots, 10n\}$ (either as cash assets or from a payment by $m_6^A$ at time $t$) then it must be the case that $m_7^A$ services the debt to $m_8^A$ at time $t$, €1 is lost to the sink and $m_7^A$ retains $x - 1$ in cash assets. At time $t + 1$, $m_{12}^A$ must service its debt to $m_7^A$ so as to replace the lost €1, with €$x$ residing at $m_7^A$ in cash assets at time $t + 1$. Consequently, there can only be at most $\frac{n}{2}$ times in $\{10, 20, \ldots, 10n\}$ when $m_7^A$ receives a payment from $m_6^A$ (recall that $m_6^A$ only makes payments to $m_7^A$ at times from $\{10, 20, \ldots, 10n\}$). When $m_7^A$ either has no cash assets or receives no payment from $m_6^A$ at time $t \in \{10, 20, \ldots, 10n\}$, we either lose €1 of cash assets from $m_{12}^A$ to the sink (and so we also lose some capacity to 'replace' resource emanating from $s$ that is lost to the sink) or we are in case (4) above and have a payment cycle involving $m_7^A$, $m_8^A$, $m_9^A$, $m_{10}^A$, $m_{11}^A$ and $m_{12}^A$. Analogous comments can be made as regards the corresponding nodes superscripted $B$ and times in $\{15, 25, \ldots, 10n + 5\}$.

Bearing in mind that none of the resource emanating from $s$ goes to the sink before it reaches either $m_6^A$ or $m_6^B$, at least $n$ distinct payments are made from $s$ and these payments result in at least $n$ payments in total from $m_6^A$ to $m_7^A$ or from $m_6^B$ to $m_7^B$. Thus, from above, $m_6^A$ must make exactly $\frac{n}{2}$ payments to $m_7^A$ and $m_6^B$ must make exactly $\frac{n}{2}$ payments to $m_7^B$. This means that any payment from $m_6^A$ to $m_7^A$ or from $m_6^B$ to $m_7^B$ must be for an amount from $\{a_1, a_2, \ldots, a_n\}$ and we have a partition of $\{a_1, a_2, \ldots, a_n\}$ into equal-sized sets both of whose sum is $k$; that is, our instance $S$ of EQUAL CARDINALITY PARTITION is a yes-instance and the claim follows. □

**Claim 2.8.2.** If $S$ is a yes-instance of EQUAL CARDINALITY PARTITION then $((G, D, A^0), 16)$ is a yes-instance of BANKRUPTCY MINIMIZATION.

*Proof.* Suppose that our instance $S = \{a_1, a_2, \ldots, a_n\}$ of EQUAL CARDINALITY PAR-

TITION is such that $n = 2m$ and $\sum_{i=1}^{m} a_{\alpha_i} = \sum_{i=1}^{m} a_{\beta_i}$, where $\{\alpha_i, \beta_i : 1 \leq i \leq m\} = \{1, 2, \ldots, n\}$. We need to build a valid schedule $\sigma$ for $(G, D, A^0)$ with at most 16 bankruptcies. Let $1 \leq i \leq n$ and suppose that $i = \alpha_j$, where $1 \leq j \leq m$. The node $s$ pays its $i$th debt to $m_1$ (that is, the debt $a_i@[10i, 10i+5]$) at time $10i$ and this payment is percolated all the way down to $m_7^A$ at time $10i$. The debt $1@10i$ from $m_7^A$ to $m_8^A$ is paid at time $10i$ with this €1 being replaced from $m_{12}^A$ at time $10i+1$ (see Case $(a.i)$ from the proof of Claim 2.8.1 above). We also have a payment cycle involving $m_8^A$, $m_9^A$, $m_{10}^A$ and $m_{11}^A$ at time $10i+1$. The cash assets of $a_i$ at $m_7^A$ are percolated down to $m_{16}^A$ at time $10i+2$. At time $10i+5$, we have suitable payment cycles involving: $m_1$, $m_2$ and $m_3$; $m_4$, $m_5$ and $m_6$; and $m_{13}^A$, $m_{14}^A$ and $m_{15}^A$. We also have a payment cycle involving $m_7^A$, $m_8^A$, $m_9^A$, $m_{10}^A$, $m_{11}^A$ and $m_{12}^A$ (see (4) from the proof of Claim 2.8.1 above). An analogous course of action is taken if $i = \beta_j$, for some $1 \leq j \leq m$. The resulting schedule is valid and the 16 nodes $m_1$, $m_3$, $m_4^A$, $m_6^A$, $m_7^A$, $m_8^A$, $m_{11}^A$, $m_{13}^A$, $m_{15}^A$, $m_4^B$, $m_6^B$, $m_7^B$, $m_8^B$, $m_{11}^B$, $m_{13}^B$ and $m_{15}^B$ are bankrupt. The claim follows. □

So, our main result holds for the PP variant of BANKRUPTCY MINIMIZATION. Note that everything above holds for the AoN variant too, though Theorem 2.12 is a strictly stronger result for that setting.

Let us consider now the FP variant. As it happens, an argument similar to that above works within the FP variant although there are more complicated nuances. Rather than repeat the whole nuanced argument in detail, and given the above complete proof for the PP variant, we only sketch the proof for the FP variant. Henceforth, we assume that we are working within the FP variant.

Consider the proof of the corresponding version of Claim 2.8.1. The reasoning that establishes that we must have that either $m_7^A$ or $m_{12}^A$ is bankrupt and that either $m_7^B$ or $m_{12}^B$ is bankrupt holds for the FP variant. Consider payments made from $m_6^A$ to $m_7^A$ at time 10.

Case $(a)$: An amount of €$x > 0$ is paid from $m_6^A$ to $m_7^A$ at time 10.

There are two essential sub-cases at time 10:

$(i)$ $m_7^A$ services its debt to $m_8^A$, which pays €1 to the sink, with perhaps €$y \geq 0$ paid from $m_7^A$ to $m_{13}^A$ and from there to the sink, so that €$x - y - 1 \geq 0$ resides at $m_7^A$ in cash assets at time 10

$(ii)$ $m_7^A$ does not fully service its debt to $m_8^A$ but pays €$w$, where $0 \leq w < 1$, to $m_8^A$, which immediately goes to the sink, and €$x - w$ to $m_{13}^A$, which immediately goes to the sink, so that €0 resides at $m_7^A$ in cash assets at time 10; hence, $m_7^A$ is bankrupt.

Consider what happens at time 11. In Case $(a.i)$, there must be a payment cycle involving $m_8^A$, $m_9^A$, $m_{10}^A$ and $m_{11}^A$ and $m_{12}^A$ services its debt to $m_7^A$. The €1 from $m_{12}^A$ does not supplement the resource emanating from $s$ but just 'replaces' €1 which was 'lost' to the sink at time 10.

Suppose that we are in Case $(a.ii)$. There are two scenarios:

(1) $m_{12}^A$ pays €1 to $m_7^A$ which pays €$1-w$ to $m_8^A$ of which €$w'$ goes immediately to the sink and €$1-w-w'$ is paid to $m_9^A$; $m_7^A$ has cash assets of at most €$w$ as it may be the case that $m_7^A$ also makes a payment to $m_{13}^A$ which goes straight to the sink, or

(2) $m_{12}^A$ pays €1 to $m_7^A$ which pays €$y$ to $m_8^A$, where $0 \leq y < 1-w$, of which €$y'$ goes immediately to the sink and €$y-y'$ is paid to $m_9^A$; also, €$1-y$ is paid to $m_{13}^A$ which goes straight to the sink.

In (1), it must be the case that $m_8^A$ receives at least $w+w'$ from $m_{11}^A$ (so as to fully service its debt to $m_9^A$); hence, $m_{11}^A$ pays at most €$1-w-w'$ to $m_{12}^A$. In any case, €$x$ have been lost to the sink with $m_7^A$ gaining €$w$ from $m_{12}^A$ (with $x \geq w$). In (2), it must be the case that $m_8^A$ receives at least €$1-(y-y')$ from $m_{11}^A$ (so as to fully service its debt to $m_9^A$); hence, $m_{11}^A$ pays at most €$y-y'$ to $m_{12}^A$. In any case, €$x$ have been lost to the sink with $m_7^A$ gaining nothing from $m_{12}^A$.

Case $(b)$: No payment is made from $m_6^A$ to $m_7^A$ at time 10.

At time 11, it must be the case that $m_{12}^A$ services its debt to $m_7^A$ and then we are essentially in Case $(a.ii)$ above.

The outcome is that the resources emanating from $m_{12}^A$ cannot supplement the resources emanating form $s$ at any time $t < 15$. An identical argument can be applied to time 15 so as to yield a similar conclusion as regards the resources emanating from $m_{12}^B$ at any time $t < 20$. The rest of the proof of Claim 2.8.1 holds for the FP variant and we have that Claim 2.8.1 holds for the FP variant.

The schedule $\sigma$ described in the proof of Claim 2.8.2 is a valid schedule in the FP variant and so Claim 2.8.2 also holds for the FP variant. This complete our proof of the main theorem. □

The proof of Theorem 2.8 clearly demonstrates the intricacies of reasoning in our financial networks. By Theorem 2.8, it follows that each of the AoN, PP and FP variants of BANKRUPTCY MINIMIZATION are *para-NP-hard* when parameterized by any parameter that is upper-bounded by the number of vertices, such as, e.g., the number of bankruptcies $k$ or the treewidth of the footprint. Note that Theorem 2.8 concerns weak completeness results (in particular, the integers in an instance of EQUAL CARDINALITY MATCHING appear explicitly as monetary debts in the corresponding instance of BANKRUPTCY MINIMIZATION).

### 2.3.2 Hardness results for PERFECT SCHEDULING

We now turn to PERFECT SCHEDULING. Since this is a subproblem of BANKRUPTCY MINIMIZATION and BAILOUT MINIMIZATION, hardness results in this section also apply to both of those problems.

**Theorem 2.9.** The problem PERFECT SCHEDULING is NP-complete for the AoN and PP variants even when we restrict to IDM instances $(G, D, A^0)$ for which: $T \leq 3$; $G$ is a directed acyclic graph with out-degree at most 3 and in-degree at most 3; the monetary amount of any debt is at most €2; and any initial cash assets of a node are at most €3 per bank.

*Proof.* Let us work within the PP variant until further notice. We introduce the *multiplier gadget* shown in Fig. 2.8 and use this gadget in our main reduction (the gadget sits within the blue dotted line). We claim the following.

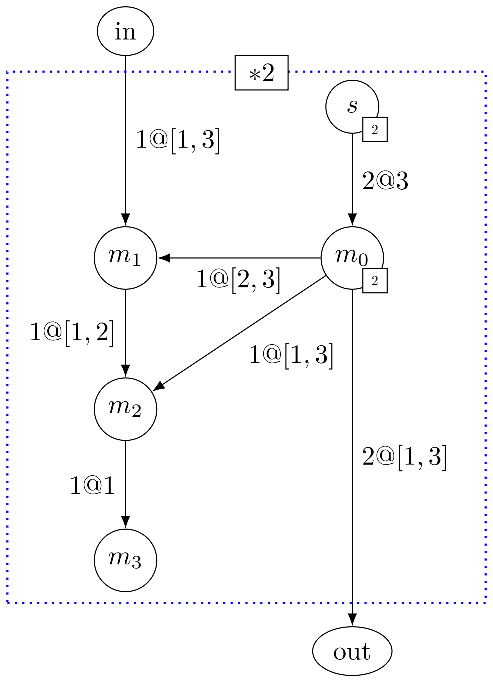


Figure 2.8: The multiplier gadget. Intuitively, the gadget "amplifies" payments into it at time 1 by a factor 2.

**Claim 2.9.1.** Assume that the node *in* of the multiplier gadget has initial cash assets of €1.

($a$) In any perfect schedule for the multiplier gadget, if no payment is made by *in* to $m_1$ at either time 1 or time 2 then no payment is made by $m_0$ to *out* at time 1.

($b$) There is a perfect schedule $\sigma_0$ for the multiplier gadget so that a payment is made by *in* at time 3.

($c$) There is a perfect schedule $\sigma_1$ for the multiplier gadget so that a payment is made by *in* to $m_1$ at time 1 and a payment of €2 is made from $m_0$ to *out* at time 1.

*Proof.* Suppose that no payment is made by *in* to $m_1$ at times 1 and 2; so $m_1$ receives no payment at time 1 and makes no payment at time 1. As there is a payment of €1 from $m_2$ to $m_3$ at time 1, there must be a payment of €1 from $m_0$ to $m_2$ at time 1. Suppose that a payment of €1 is made from $m_0$ to *out* at time 1. If so then $m_0$ can make no payment to $m_1$ at time 2 and $m_1$ is bankrupt which yields a contradiction. Hence, there is no payment from $m_0$ to *out* at time 1. The statement ($a$) follows.

Consider the schedule whereby:

- at time 1: $m_0$ pays €1 to $m_2$; $m_2$ pays €1 to $m_3$
- at time 2: $m_0$ pays €1 to $m_1$; $m_1$ pays €1 to $m_2$
- at time 3: $s$ pays €2 to $m_0$; $m_0$ pays €2 to *out*; *in* pays €1 to $m_1$.

This yields a perfect schedule and statement $(b)$ follows.

Suppose that there is a payment of €1 made from *in* to $m_1$ at time 1; so, we can also make payments of €1 from $m_1$ to $m_2$ and from $m_2$ to $m_3$ at time 1. Additionally, we can make a payment of €2 from $m_0$ to *out* at time 1. At time 2, no payments are made. At time 3, we can make payments of: €2 from $s$ to $m_0$; €1 from $m_0$ to $m_2$; and €1 from $m_0$ to $m_1$. This yields a perfect schedule and statement $(c)$ follows. □

Our reduction is from 3-SAT-3 (again) to PERFECT SCHEDULING. As before, we assume that all 3-SAT-3 instances are such that every literal appears at least once and at most twice and that no clause contains both a literal and its negation. Given a 3-SAT-3 instance $\phi$ involving $n$ Boolean variables and $m$ clauses, we construct an IDM instance $(G, D, A^0)$ as portrayed in Fig. 2.9 (we omit the formal description of $(G, D, A^0)$ as it can be immediately derived from Fig. 2.9; moreover, we proceed similarly with other IDM instances that we construct later on). The types of nodes (source, literal, clause and sink) are as in the proof of Theorem 2.7, although we have additional so-called $a$-type nodes, and we abbreviate our multiplier gadget as a square box with $*2$ inside (note that we have $2n$ distinct copies of our multiplier gadget where the node *in* is taken as $a_i$ and the node *out* as the literal node $x_i$ or the literal node $\neg x_i$, for $1 \leq i \leq n$; of course, we have $m$ clause nodes, $n$ source nodes, $n$ $a$-type nodes and one sink node).

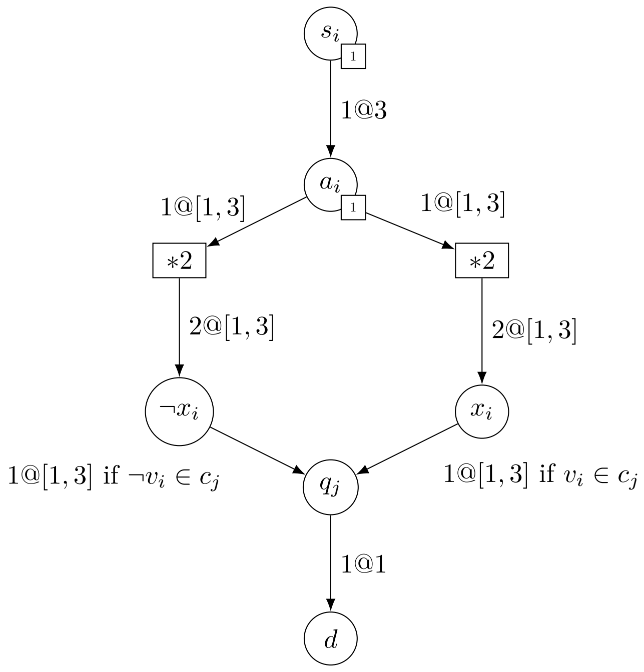


Figure 2.9: Illustration of the reduction from 3-SAT 3 to PERFECT SCHEDULING restricted to Directed Acyclic Graphs (DAGs).

**Claim 2.9.2.** If $(G, D, A^0)$ is a yes-instance of PERFECT SCHEDULING then $\phi$ is a yes-instance of 3-SAT-3.

*Proof.* Define the truth assignment $X$ via: if, within $\sigma$, $x_i$ receives a payment of at least €1 at time 1 then $X(v_i) = True$; otherwise $X(v_i) = False$.

Fix $1 \leq j \leq m$. We must have that $q_j$ pays €1 to $d$ at time 1 and so $q_j$ must receive €1 from some node $x_i$ at time 1 or €1 from some node $\neg x_i$ at time 1.

Suppose that $q_j$ receives €1 from $x_{\tau_j}$ at time 1; in particular, $v_{\tau_j} \in c_j$. Hence, $x_{\tau_j}$ receives at least €1 from its corresponding multiplier gadget at time 1 and so, by definition, $X(v_{\tau_j}) = True$ with the clause $c_j$ satisfied by $X$.

Alternatively, suppose that $q_j$ receives €1 from $\neg x_{\tau_j}$ at time 1; in particular, $\neg v_{\tau_j} \in c_j$. Hence, $\neg x_{\tau_j}$ receives at least €1 from its corresponding multiplier gadget at time 1. By Claim 2.9.1.$a$, there must be a payment of €1 made from $a_{\tau_j}$ to this multiplier gadget at either time 1 or time 2. Consequently, no payment is made by $a_{\tau_j}$ to the complementary multiplier gadget (that is, the one with a debt to $x_{\tau_j}$) at either time 1 or time 2. By Claim 2.9.1.$a$, no payment is received by $x_i$ at time 1 and so, by definition, $X(v_{\tau_j}) = False$ with the clause $c_j$ satisfied by $X$. The claim follows. □

**Claim 2.9.3.** If $\phi$ is a yes-instance of 3-SAT-3 then $(G, D, A^0)$ is a yes-instance of PERFECT SCHEDULING.

*Proof.* Let $X$ be a satisfying truth assignment for $\phi$. For each clause $c_j$, let $v_{\tau_j}$ be a Boolean variable whose occurrence in $c_j$, either via the literal $v_{\tau_j}$ or the literal $\neg v_{\tau_j}$, leads to $c_j$ being satisfiable. So, we get a list $L = v_{\tau_1}, v_{\tau_2}, \ldots, v_{\tau_m}$ of 'satisfying' Boolean variables, possibly with repetitions although no variable appears in the list more than twice and if a Boolean variable $v$ does appear twice then the corresponding literals in the two corresponding clauses are both positive or both negative (of course, this stems from the format of $\phi$ as an instance of 3-SAT-3): if the occurrences are positive then we say that $v$ has *positive polarity*, with *negative polarity* defined analogously. Note that any debt from a literal node to a clause node in our IDM instance exists solely because of the occurrence of the corresponding literal in the corresponding clause; in particular, we can never have debts from both literal nodes corresponding to some variable to the same clause node.

Consider the following schedule $\sigma$. At time 1, for each $1 \leq j \leq m$:

- $q_j$ pays €1 to $d$
- if $v_{\tau_j}$ appears in $L$ with positive (resp. negative) polarity then $x_{\tau_j}$ (resp. $\neg x_{\tau_j}$) pays €1 to $q_j$
- if $v_{\tau_j}$ appears in $L$ with positive (resp. negative) polarity then $a_{\tau_j}$ pays €1 to the multiplier gadget which has a debt to $x_{\tau_j}$ (resp. $\neg x_{\tau_j}$).

In addition, for each $1 \leq j \leq m$, if $v_{\tau_j}$ appears in $L$ with positive (resp. negative) polarity then:

- within the multiplier gadget that has a debt to $x_{\tau_j}$ (resp. $\neg x_{\tau_j}$), we include the schedule $\sigma_1$ from Claim 2.9.1.$c$

- within the multiplier gadget that has a debt to $\neg x_{\tau_j}$ (resp. $x_{\tau_j}$), we include the schedule $\sigma_0$ from Claim 2.9.1.*b*.

At time 3, for each $1 \leq j \leq m$:

- $s_{\tau_j}$ pays €1 to $a_{\tau_j}$
- if $v_{\tau_j}$ appears in $L$ with positive (resp. negative) polarity then $a_{\tau_j}$ pays €1 to the multiplier gadget which has a debt to $\neg x_{\tau_j}$ (resp. $x_{\tau_j}$).

Finally, having done the above, add any Boolean variable $v$ not appearing in $L$ to $L$ and proceed as above with these Boolean variables and assuming that they have positive polarity (note that the restriction of $X$ to these Boolean variables has no effect on whether $X$ satisfies $\phi$). What results is a perfect schedule. □

Given that our IDM instance $(G, D, A^0)$ of Perfect Scheduling can be built from the instance $\phi$ of 3-Sat-3 in polynomial time, we obtain our result for the PP variant.

Consider now the AoN variant. As it happens, all of the above schedules are perfect schedules within the AoN variant and all associated reasoning still holds. Hence, we have our result for the AoN variant too. □

**Theorem 2.10.** The problem Perfect Scheduling is NP-complete for the PP and AoN variants even when we restrict to IDM instances $(G, D, A^0)$ for which: $G$ is a multiditree with diameter 6; all debts have monetary amount €1; and there is a maximum of 6 debts between any pair of nodes.

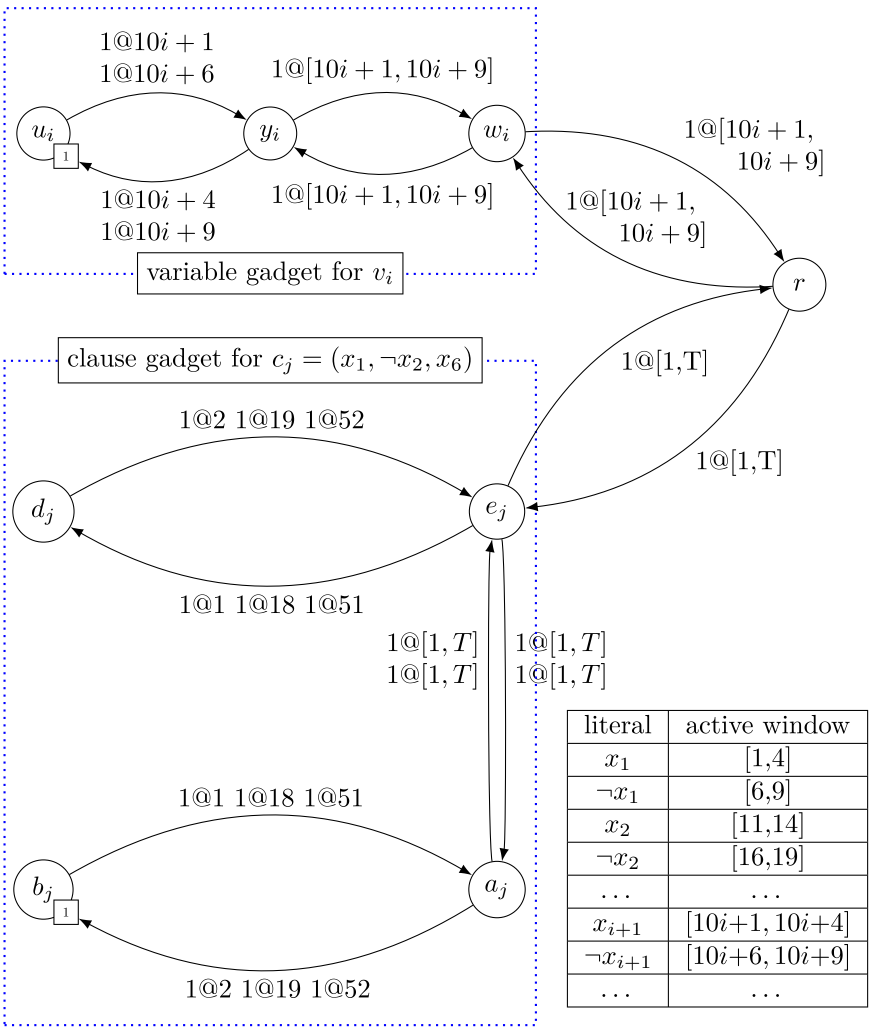
*Proof.* We show that, given an instance $\phi$ of 3-Sat-3, involving $n$ Boolean variables and $m$ clauses, we can construct in polynomial-time an IDM instance $(G, D, A^0)$ where $G$ satisfies the criteria stated in the theorem so that $(G, D, A^0)$ admits a perfect schedule iff $\phi$ has a satisfying truth assignment. As usual, we restrict ourselves to those instances of 3-Sat-3 in which every literal appears at least once and at most twice and where no clause contains both a literal and its negation. In particular, we can label any appearance of any literal in $\phi$ as the first appearance or the second appearance.

Our reduction is portrayed in Fig. 2.10. There is a distinct *variable gadget* for each Boolean variable $v_i$, with $1 \leq i \leq n$, and a distinct *clause gadget*, for each clause $c_j$, with $1 \leq j \leq m$. There is one node $r$. The debts in any variable gadget or involving the node $r$ are self-evident whereas the debts in a clause gadget are more involved.

- If the literal $v_i$ is in the clause $c_j$ and this appearance is the first (resp. second) appearance of $v_i$ in $\phi$ then there are:
  - debts $1@10(i-1)+1$ (resp. $1@10(i-1)+3$) from $b_j$ to $a_j$ and from $e_j$ to $d_j$
  - debts $1@10(i-1)+2$ (resp. $1@10(i-1)+4$) from $a_j$ to $b_j$ and from $d_j$ to $e_j$.

- If the literal $\neg v_i$ is in the clause $c_j$ and this appearance is the first (resp. second) appearance of $\neg v_i$ in $\phi$ then there are:
  - debts $1@10(i-1)+6$ (resp. $1@10(i-1)+8$) from $b_j$ to $a_j$ and from $e_j$ to $d_j$
  - debts $1@10(i-1)+7$ (resp. $1@10(i-1)+9$) from $a_j$ to $b_j$ and from $d_j$ to $e_j$.
- If the clause $c_j$ has 3 literals (resp. 2 literals) then there are:
  - two separate[1] debts $1@[1,T]$ (resp. a single debt $1@[1,T]$) from $a_j$ to $e_j$ and from $e_j$ to $a_j$.

The legend in Fig. 2.10 shows the time intervals corresponding to each literal, which we call the *active windows* of the literals, and the occurrence of $x_1$ (resp. $\neg x_2$, $x_6$) in $c_j$ in Fig. 2.10 is the first (resp. second, first) occurrence.



| literal | active window |
|---|---|
| $x_1$ | [1,4] |
| $\neg x_1$ | [6,9] |
| $x_2$ | [11,14] |
| $\neg x_2$ | [16,19] |
| ... | ... |
| $x_{i+1}$ | $[10i{+}1, 10i{+}4]$ |
| $\neg x_{i+1}$ | $[10i{+}6, 10i{+}9]$ |
| ... | ... |

Figure 2.10: A reduction from 3-SAT-3 to PERFECT SCHEDULING restricted to multiditrees.

**Claim 2.10.1.** If $(G, D, A^0)$ is a yes-instance of PERFECT SCHEDULING then $\phi$ is a yes-instance of 3-SAT-3.

[1]By having only unit debts in the instance we have that every PP schedule is also an AoN schedule, and vice versa; for the PP variant we could instead have a single debt $2@[1,T]$.

*Proof.* Suppose that there is a perfect schedule $\sigma$ for $(G, D, A^0)$. Consider a clause gadget, corresponding to the clause $c_j$. There are 4 debts due within each active window corresponding to a literal in the clause. Suppose that the active windows are $[\alpha_1, \beta_1]$, $[\alpha_2, \beta_2]$ and $[\alpha_3, \beta_3]$, with $\beta_1 < \alpha_2$ and $\beta_2 < \alpha_3$ (we are assuming that our clause has 3 literals but the arguments for clauses with only 2 literals run analogously). If no payment has been received by $e_j$ from $r$ by time $\beta_1 + 1$ then: the €1 at $b_j$ within the clause gadget must have been used in $\sigma$:

- to pay the debts of $b_j$ to $a_j$, $a_j$ to $e_j$ and $e_j$ to $d_j$ at time $\alpha_1$
- to pay the debts of $d_j$ to $e_j$, $e_j$ to $a_j$ and $a_j$ to $b_j$ at time $\alpha_1 + 1$,

if the appearance of the corresponding literal is the first; or

- to pay the debts of $b_j$ to $a_j$, $a_j$ to $e_j$ and $e_j$ to $d_j$ at time $\alpha_1 + 2$
- to pay the debts of $d_j$ to $e_j$, $e_j$ to $a_j$ and $a_j$ to $b_j$ at time $\alpha_1 + 3 = \beta_1$,

if the appearance of the corresponding literal is the second (note that the payments made towards the debts of $a_j$ to $e_j$ and $e_j$ to $a_j$ are only partial payments). We have an analogous situation as regards the interval $[\alpha_2, \beta_2]$ when no payment has been received by $e_j$ from $r$ by time $\beta_2 + 1$. However, if no payment has been received by $e_j$ from $r$ by time $\beta_3 + 1$ then we obtain a contradiction as the debts from $a_j$ to $e_j$ and from $e_j$ to $a_j$ will have been fully paid and consequently $e_j$ will be bankrupt at time $\beta_3$. Hence, within $\sigma$, there must be a payment of €1 received by $e_j$ from $r$ and this is the only payment made from $r$ to $e_j$. Furthermore, there can be no payment from $e_j$ to $r$ strictly before the time at which a payment is made from $r$ to $e_j$ and if a payment is made from $e_j$ to $r$ at the same time that a payment is made from $r$ to $e_j$ then this can be interpreted as no resource leaving or entering the clause gadget, with the external resource satisfying both debts involving $r$ and $e_j$, which cannot be the case.

Consider now a variable gadget, corresponding to the variable $v_i$. At times $t \in [0, 10i] \cup [10i + 4, 10i + 5] \cup [10i + 9, T]$ there must be cash assets of (at least) €1 at node $u_i$. So, outside the active windows of the literals associated with the variable $v_i$, there must be at least €1 of cash assets at the node $u_i$. Also, note that the €1 originating at node $u_i$ can only leave and return to the variable gadget at some times in $[10i + 1, 10i + 5]$ or in $[10i + 6, 10i + 9]$ but not both. Consequently, at any time, there is at most €1 of resource that originated within a variable gadget outside that particular variable gadget.

Consider the time interval $[1, 9]$. Within this time interval, the €1 originating at $u_1$ is the only euro that might be possibly 'outside' the variable gadget corresponding to $v_1$. Call this euro $E$. Suppose $E$ leaves its variable gadget at some time in $[1, 4]$. It needs to be back at $y_1$ by time 4 (and will never leave the variable gadget again). Suppose that $E$ is paid from $r$ to $e_j$, for some $j$, within the time interval $[1, 4]$. If the literal $x_1$ is not in $c_j$ then all debts involving only $a_j$ and $b_j$ and all debts involving only $d_j$ and $e_j$ will be due at some time outside $[1, 4]$ and so $E$ is of no use to the clause gadget for $c_j$. If $E$ is paid from $r$ to $e_j$ at time 1 (resp. by time 3) and the literal $x_1$ is in clause $c_j$ as the first (resp. second) appearance then $E$ can be used to pay the debts from $e_j$ to $d_j$ at time 1 (resp. time 3) and from $d_j$ to $e_j$ at time 2 (resp. time 4). Note that $E$ cannot be used to pay the debt from $a_j$ to $b_j$ at time 2 or time 4 as it would not get back to its variable gadget by time 4. Given that $E$ leaves the clause gadget by time 4, no more resource either enters or leaves the clause gadget. Alternatively, suppose that $E$ leaves the variable gadget corresponding to $v_1$ at some time in $[6, 9]$. Exactly the same argument can be made as that above except with respect to the literal $\neg x_1$ appearing in some clause or other.

Let us continue with the time interval $[11, 19]$ and the €1 originating at $u_2$. An analogous argument to that above holds. Note that all clause gadgets that were previously 'visited' by $E$, above, are now 'closed' in that they accept or eject no further resource. Indeed, an analogous argument to that above holds for every euro originating at some node $u_i$. As $\sigma$ is a perfect schedule, every clause gadget must be visited by some euro originating in some variable gadget and the particular literal corresponding to the active window during which the euro left its variable gadget must appear in the clause. Define the truth assignment $X$ via: $X(v_i) = True$ (resp. $False$) if the euro from the variable gadget corresponding to $v_i$ leaves its variable gadget during the active window $[10i + 1, 10i + 4]$ (resp. $[10i + 6, 10i + 9]$) and visits a clause gadget, with any Boolean variables $v_i$ for which $X(v_i)$ has not been defined such that $X(v_i)$ is defined arbitrarily. Given the above discussion, it should be clear that $X$ satisfies $\phi$. $\square$

**Claim 2.10.2.** If $\phi$ is a yes-instance of 3-SAT-3 then $(G, D, A^0)$ is a yes-instance of PERFECT SCHEDULING.

*Proof.* Suppose that $X$ is a satisfying truth assignment for $\phi$. Define the schedule $\sigma$ as follows.

If $X(v_i) = True$ then at time $10(i - 1) + 1$, the euro originating at $u_i$ is paid from $u_i$ to $y_i$ to $w_i$ to $r$.

- If the clause $c_j$ containing the first appearance of the literal $x_i$ exists and has not been visited by some euro originating within a variable gadget then at time $10i + 1$, the euro is paid from $r$ to $e_j$ and on to $d_j$. At time $10i + 2$, the same euro is paid from $d_j$ to $e_j$ and on to $r$.
- If there is no clause containing the literal $x_i$ or if the clause gadget corresponding to the first appearance of $x_i$ has already been visited in the schedule $\sigma$ by some euro originating within a variable gadget, then do nothing.
- If the clause $c_j$ containing the second appearance of the literal $x_i$ exists and has not been visited by some euro originating within a variable gadget then at time $10i + 3$, the euro is paid from $r$ to $e_j$ and on to $d_j$. At time $10i + 4$, the same euro is paid from $d_j$ to $e_j$ and on to $r$.
- If there is no clause containing the literal $x_i$ or if there is no second appearance of the literal $x_i$ or if the clause gadget corresponding to the second appearance

of $x_i$ has already been visited in the schedule $\sigma$ by some euro originating within a variable gadget, then do nothing.

- Our euro at $r$ is paid at time $10i + 4$ from $r$ to $w_i$ to $y_i$ to $u_i$, and is then available to pass back from $u_i$ to $y_i$ at time $10i + 6$, and back from $y_i$ to $u_i$ at time $10i + 9$ – without exiting the gadget.

If $X(v_i) = False$ then at time $10i+6$, the euro originating at $u_i$ is paid from $u_i$ to $y_i$ to $w_i$ to $r$. (Prior to this, the euro originating at $u_i$ passes to $y_i$ at time $10i + 1$ and back to $u_i$ at time $10i + 4$ – again without exiting the gadget.)

- If the clause $c_j$ containing the first appearance of the literal $\neg x_i$ exists and has not been visited by some euro originating within a variable gadget then at time $10i + 6$, the euro is paid from $r$ to $e_j$ and on to $d_j$. At time $10i + 7$, the same euro is paid from $d_j$ to $e_j$ and on to $r$.
- If there is no clause containing the literal $\neg x_i$ or if the clause gadget corresponding to the first appearance of $\neg x_i$ has already been visited in the schedule $\sigma$ by some euro originating within a variable gadget, then do nothing.
- If the clause $c_j$ containing the second appearance of the literal $\neg x_i$ exists and has not been visited by some euro originating within a variable gadget then at time $10i + 8$, the euro is paid from $r$ to $e_j$ and on to $d_j$. At time $10i + 9$, the same euro is paid from $d_j$ to $e_j$ and on to $r$.
- If there is no clause containing the literal $\neg x_i$ or if there is no second appearance of the literal $\neg x_i$ or if the clause gadget corresponding to the second appearance of $\neg x_i$ has already been visited in the schedule $\sigma$ by some euro originating within a variable gadget, then do nothing.
- Our euro at $r$ is paid at time $10i + 9$ from $r$ to $w_i$ to $y_i$ to $u_i$.

Within any clause gadget corresponding to $c_j$, the euro originating at $b_j$ is used to pay the debts corresponding to the literal not addressed by the euro from a variable gadget. It can easily be seen that $\sigma$ is valid and a perfect schedule. □

The result follows, given that the construction of $(G, D, A^0)$ can clearly be undertaken in polynomial-time. □

In all the above results, the input IDM instance is allowed to have unlimited (i.e., unbounded) total initial assets which might be unrealistic in practically relevant financial systems. We now show that even in the highly restricted case where there is just €1 in initial external assets in total, PERFECT SCHEDULING still remains NP-complete in the AoN and PP variants.

**Theorem 2.11.** The problem PERFECT SCHEDULING is NP-complete in the AoN and PP variants even when the total value of all initial external assets in any instance is €1.

*Proof.* The following proof applies to both the AoN and PP variants. Our reduction is a reduction from the problem SOURCED HAMILTONIAN PATH defined as follows.

SOURCED HAMILTONIAN PATH

Instance: a digraph $H$ and a vertex $x$

Yes-instance: there exists a Hamiltonian path in $H$ with source $x$.

This problem can be trivially shown to be NP-complete by reducing from the standard problem of deciding whether a digraph has a Hamiltonian cycle [10].

Let $H$ be some digraph on the $n$ vertices $\{x_i : 1 \leq i \leq n\}$ and w.l.o.g. let $x = x_1$. In order to describe our reduction, we first describe a gadget, namely the *at-least-once* gadget. We have one of these gadgets for each vertex of $H$ and we refer to the gadget corresponding to the vertex $x_i$ of $H$ as at-least-once($i$). Our at-least-once gadget can be defined as in Fig. 2.11 where the value $T$ is defined as $2n+1$. Note that the gadget is exactly the nodes and debts within the blue dotted box and so contains its own copies of nodes $v_L$, $v_C$ and $v_R$ and the $4n-1$ debts involving them. The nodes $v'_R$ and $v''_L$ are not part of the gadget but are nodes in other gadgets as we now explain.

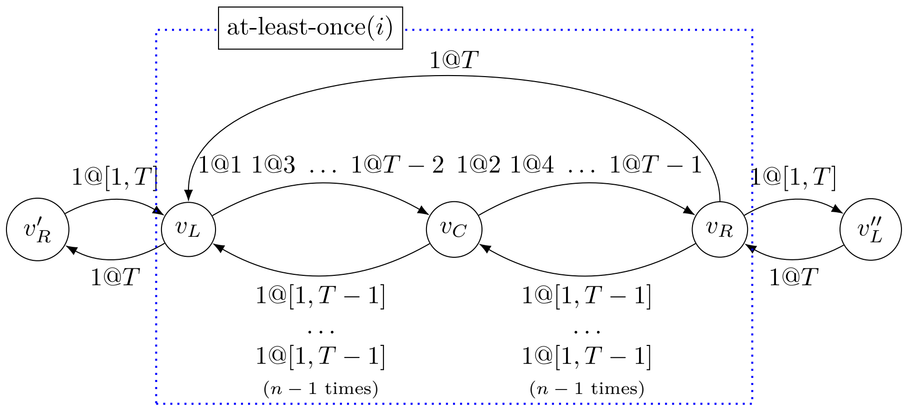


Figure 2.11: An at-least-once gadget. Note that as $T = 2n+1$ there are, $n$ €1 debts owed by $v_L$ to $v_C$ and by $v_C$ to $v_R$.

Set $T = 2n+1$. Our IDM instance $(G, D, A^0)$ can be defined as follows:

- there is the at-least-once($i$) gadget, for $1 \leq i \leq n$
- for every edge $(x_i, x_j)$ of $H$, there is a debt of €1 from $v_R$ of at-least-once($i$) to $v_L$ of at-least-once($j$) to be paid in the interval $[1, T]$ and a debt of €1 from $v_L$ of at-least-once($j$) to $v_R$ of at-least-once($i$) to be paid at time $T$
- all nodes have initial external assets of 0 except for node $v_L$ of at-least-once(1) which has initial external assets of 1.

We refer to the single euro of initial external assets as the *initial euro*. The IDM instance $(G, D, A^0)$ can clearly be constructed in time polynomial in $n$.

**Claim 2.11.1.** If $(G, D, A^0)$ is a yes-instance of PERFECT SCHEDULING then $(H, x)$ is a yes-instance of SOURCED HAMILTONIAN PATH.

*Proof.* Note that strictly prior to time $T$, the only payment-cycles that can exist within $G$ involve the two nodes $v_L$ and $v_C$ of some at-least-once gadget or the two nodes $v_C$ and $v_R$ of some at-least-once gadget. Note also that within some gadget there are $n$ debts of €1 from $v_L$ to $v_C$ needing to be satisfied and $n-1$ debts of €1 from $v_C$ to $v_L$. An analogous statement can be made as regards $v_C$ and $v_R$. Consequently, in order to satisfy all debts involving $v_L$ and $v_C$ within some gadget, at some odd time in $[1, T-1]$, the initial euro must be within that gadget so as to satisfy some debt from $v_L$ to $v_C$ (by moving from $v_L$ to $v_C$). Similarly, at some even time in $[1, T-1]$, the initial euro must be within the gadget so as to satisfy some debt from $v_C$ to $v_R$ (by moving from $v_C$ to $v_R$). Moreover, at any time in $[1, T-1]$, the initial euro can only satisfy at most one of the debts mentioned above. Hence, given that $T = 2n+1$, at any time in $[1, T-1]$, the initial euro must be satisfying exactly one of the above debts.

Suppose that the initial euro satisfies some debt from $v_L$ to $v_C$ in some at-least-once gadget at time $t$. As the initial euro needs to satisfy one of the above debts at time $t+1$, we need that at time $t+1$ the initial euro satisfies a debt from $v_C$ to $v_R$ in the same gadget. Also, it cannot be the case that a debt from $v_C$ to $v_R$ in some gadget is satisfied by the initial euro before the euro satisfies some debt from $v_L$ to $v_R$ in the same gadget. As the initial euro starts at $v_L$ in at-least-once(1), our schedule must be such that the initial euro 'moves' through the at-least-once gadgets corresponding to every node, entering at the node $v_L$ and exiting at the node $v_R$. Consequently, its path within $G$ corresponds to a path $x = x_1, x_2, \ldots, x_n$ in $H$ where every node on this path is distinct and where there is a directed edge from node $x_i$ to $x_{i+1}$, for $1 \leq i \leq n-1$; that is, a Hamiltonian path in $H$ with source $x$. □

**Claim 2.11.2.** If $(H, x)$ is a yes-instance of Sourced Hamiltonian Path then $(G, D, A^0)$ is a yes-instance of Perfect Scheduling.

*Proof.* Let $x = x_1, x_2, \ldots, x_n$ be a Hamiltonian path $P$ in the digraph $H$. Consider the following schedule $\sigma$:

- the initial euro is used so as to pay the following debts:
  - €1 at time $2i-1$ from $v_L$ to $v_C$ in at-least-once($x_i$) and €1 at time $2i$ from $v_C$ to $v_R$ in at-least-once($x_i$), for $1 \leq i \leq n$
  - €1 at time $2i$ from $v_R$ in at-least-once($x_i$) to $v_L$ in at-least-once ($x_{i+1}$), for $1 \leq i \leq n-1$
- for any $v_L$ and $v_C$ within some at-least-once gadget and at any odd time $t < T$ when a debt is not being paid from $v_L$ to $v_C$ using the initial euro, there is a payment-cycle consisting of payments of €1 from $v_L$ to $v_C$ and of €1 from $v_C$ to $v_L$
- for any $v_C$ and $v_R$ within some at-least-once gadget and at any even time $t < T$ when a debt is not being paid from $v_C$ to $v_R$ using the initial euro, there is a

payment-cycle consisting of payments of €1 from $v_C$ to $v_R$ and of €1 from $v_R$ to $v_C$

- for any edge $(x_i, x_j)$ of $H$ that does not feature in the Hamiltonian path $P$, there is a payment-cycle consisting of payments at time $T$ of €1 from $v_R$ in at-least-once($x_i$) to $v_L$ in at-least-once($x_j$) and of €1 from $v_L$ in at-least-once($x_j$) to $v_R$ in at-least-once($x_i$)
- the initial euro is used so as to pay the following debts:
  - €1 at time $T$ from $v_R$ to $v_L$ in at-least-once($x_i$), for $1 \leq i \leq n$
  - €1 at time $T$ from $v_L$ in at-least-once($x_i$) to $v_R$ in at-least-once($x_{i-1}$), for $2 \leq i \leq n$.

The 'path' taken by the initial euro within $(G, D, A^0)$ can be visualized as in Fig. 2.12 where the debt arrows are tagged with the time of payment. It is clear that $\sigma$ is valid and a perfect schedule. □

Our main result follows. □

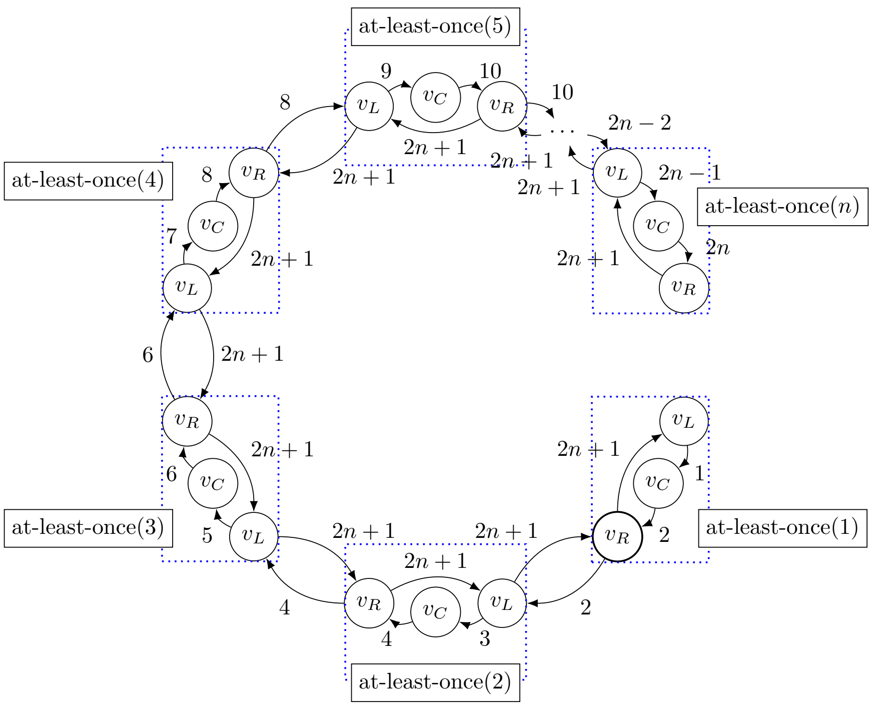


Figure 2.12: The path taken by the initial euro straightforwardly corresponds to a sourced Hamiltonian path in the original graph.

Of course, one can obtain additional restrictions on the structure of the IDM instances for Perfect Scheduling in Theorem 2.11 by looking at NP-completeness results relating to Sourced Hamiltonian Path on restricted digraphs; however, we have refrained from doing so (as nothing of any significance emerges).

We can constrain the digraph $G$ of an instance $(G, D, A^0)$ of Perfect Scheduling even further in the AoN variant; indeed, so that it is always a directed path of length 3. The price we pay is that the initial external assets are potentially large.

**Theorem 2.12.** Consider the problem Perfect Scheduling restricted so that every instance $(G, D, A^0)$ is such that $G$ is a directed path of bounded length.

($a$) If, further, $T$ is restricted to be 2 then the resulting problem is weakly NP-complete in the AoN variant.

($b$) If there are no restrictions on $T$ then the resulting problem is strongly NP-complete in the AoN variant.

*Proof.* Consider ($a$). We reduce from the problem Partition defined as follows (and proven in [10] to be weakly NP-complete).

<u>Partition</u>

Instance: a multi-set of integers $S = \{a_1, a_2, \ldots, a_n\}$ with $sum(S) = 2k$

Yes-instance: there exists a partition of $S$ into two subsets $S_1$ and $S_2$ such that $sum(S_1) = sum(S_2) = k$.

In general, an instance $S = \{a_1, a_2, \ldots, a_n\}$ has size $n\beta$ where $\beta$ is the least number of bits required to express any of the integers of $S$ in binary.

Let $S$ be an instance of Partition of size $nb$. Consider the IDM instance $(G, D, A^0)$ in Fig. 2.13. Note that the time taken to construct $(G, D, A^0)$ from $S$ is polynomial in $nb$.

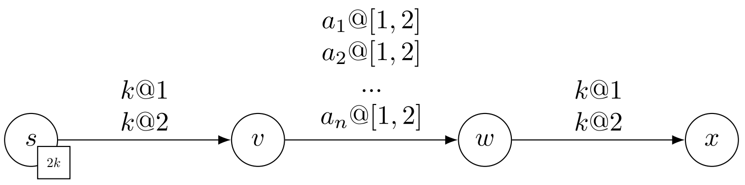


Figure 2.13: An IDM instance encoding an instance $\{a_1, \ldots, a_n\}$ of Partition with sum $2k$.

**Claim 2.12.1.** If $(G, D, A^0)$ is a yes-instance of Perfect Scheduling then $S$ is a yes-instance of Partition.

*Proof.* Suppose that there is a valid schedule $\sigma$ for $(G, D, A^0)$ that is a perfect schedule. It must be the case that the total amount paid by $s$ at time 1 is €$k$ and that this payment is immediately paid by $v$ to $w$ and from $w$ to $x$ at time 1. As we are in the AoN variant, it must be the case that the sum of a subset of integers of $S$ amounts to $k$. An analogous argument applies to the payments made at time 2 and the remainder of the integers in $S$. The claim follows. □

**Claim 2.12.2.** If $S$ is a yes-instance of Partition then $(G, D, A^0)$ is a yes-instance of Perfect Scheduling.

*Proof.* Suppose that $S_1$ and $S_2$ is a partition of $S$ such that $\text{sum}(S_1) = \text{sum}(S_2) = k$. Define the schedule $\sigma$ so that:

- at time 1: $s$ pays €$k$ to $v$; if $a_i \in S_1$, for $1 \le i \le n$, then $v$ pays €$a_i$ to $w$; and $w$ pays €$k$ to $x$

- at time 2: $s$ pays €$k$ to $v$; if $a_i \in S_2$, for $1 \leq i \leq n$, then $v$ pays €$a_i$ to $w$; and $w$ pays €$k$ to $x$.

The schedule $\sigma$ is a valid perfect schedule. □

The proof of $(a)$ follows. Now consider $(b)$. We reduce from the strongly NP-complete problem 3-PARTITION defined as follows (see [71]).

3-PARTITION

Instance: a multi-set of integers $S = \{a_1, a_2, \ldots, a_{3m}\}$, for some $m \geq 1$, with $sum(S) = mk$

Yes-instance: there exists a partition of $S$ into $m$ triplets $S_1, S_2, \ldots S_m$ such that $sum(S_i) = k$, for each $1 \leq i \leq m$.

In general, an instance $S = \{a_1, a_2, \ldots, a_{3m}\}$ has size $m\beta$ where $\beta$ is the least number of bits required to express any of the integers of $S$ in binary.

Let $S$ be an instance of 3-PARTITION of size $m\beta$. By multiplying all integers by 4 if necessary, we may assume that every integer of $S$ is divisible by 4 as is $k$. Consider the IDM instance $(G, D, A^0)$ in Fig. 2.13. Note that the time taken to construct $(G, D, A^0)$ from $S$ is polynomial in $m\beta$.

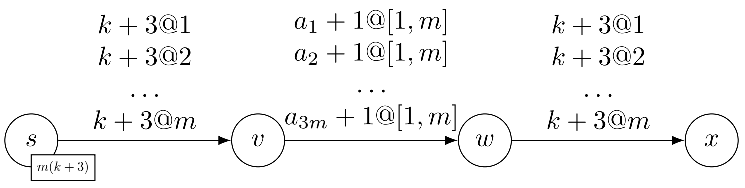


Figure 2.14: An IDM instance showing the reduction from 3-PARTITION to AoN PERFECT SCHEDULING.

**Claim 2.12.3.** If $(G, D, A^0)$ is a yes-instance of PERFECT SCHEDULING then $S$ is a yes-instance of 3-PARTITION.

*Proof.* Suppose that there is a valid schedule $\sigma$ for $(G, D, A^0)$ that is a perfect schedule. It must be the case that the total amount paid by $s$ at time $i$, for every $1 \leq i \leq m$, is €$k+3$ and that this payment is immediately paid by $v$ to $w$ and by $w$ to $x$. Suppose that the payment at time $i$ by $v$ to $w$ pays at least 4 of the debts due. So, there exists another time $j$, say, where the payments made by $v$ to $w$ pay at most 2 debts. So, we have that either $a_\alpha + a_\beta + 2 = k + 3$ or $a_\alpha + 1 = k + 3$, for some $1 \leq \alpha \neq \beta \leq 3m$. This yields a contradiction as the right-hand sides of these equations are equivalent to 3 modulo 4 whereas the left-hand sides are not. So, at any time $i$, for $1 \leq i \leq m$, exactly three debts are paid by $v$ to $w$ at time $i$. If $1 \leq \alpha, \beta, \gamma \leq 3m$ are distinct so that debts of monetary amounts $a_\alpha + 1$, $a_\beta + 1$ and $a_\gamma + 1$ are paid by $v$ to $w$ at some time then $a_\alpha + 1 + a_\beta + 1 + a_\gamma + 1 = k + 3$; that is, $a_\alpha + a_\beta + a_\gamma = k$. So, we have a yes-instance of 3-PARTITION. The claim follows. □

**Claim 2.12.4.** If $S$ is a yes-instance of 3-Partition then $(G, D, A^0)$ is a yes-instance of Perfect Scheduling.

*Proof.* Suppose that $S$ can be partitioned into triplets so that the sum of the integers in each triplet is $k$; so, suppose that $a_{\alpha_i} + a_{\beta_i} + a_{\gamma_i} = k$, for each $1 \leq i \leq m$, where $S = \{a_{\alpha_i}, a_{\beta_i}, a_{\gamma_i} : 1 \leq i \leq m\}$. Define the following schedule $\sigma$: at time $i$, for each $1 \leq i \leq m$, $s$ pays €$k+3$ to $v$; $v$ pays €$a_{\alpha_i} + a_{\beta_i} + a_{\gamma_i} + 3 = k + 3$ to $w$; and $w$ pays €$k+3$ to $x$. The schedule $\sigma$ is valid and a perfect schedule. The claim follows. □

The main result follows. □

### 2.3.3 Hardness results for Bankruptcy Maximization

We now turn to Bankruptcy Maximization.

**Theorem 2.13.** The problem Bankruptcy Maximization is NP-complete in the AoN, PP and FP variants even when for an instance $(G, D, A^0)$: $T = 2$; $G$ is a directed acyclic graph with out-degree at most 2, in-degree at most 3; all monetary debts are at most €2 per edge; and initial external assets are at most €3 per bank.

*Proof.* We build a polynomial-time reduction from the problem 3-Sat-3, with the usual restrictions on instances (see the proof of Theorem 2.7). Suppose that we have some instance $\phi$ of 3-Sat-3 where there are $n$ Boolean variables and $m$ clauses. We start with the *chain gadget* chain$(l)$, where $l \geq 1$, as portrayed in Fig. 2.15 (note that the gadget is the path of $l$ nodes within the blue dotted box). The key point about any chain gadget is that in some schedule: if at time 1, node $u$ does not make a payment to node $m_1$ then $u$ and all the nodes of the chain gadget are bankrupt; and if at time 1, $u$ pays its debt to $m_1$ then none of the nodes of the chain gadget is bankrupt. As in our proof of Theorem 2.8, we first work in the PP variant unless otherwise stated, though the reasoning will apply to the AoN and FP variants as well.

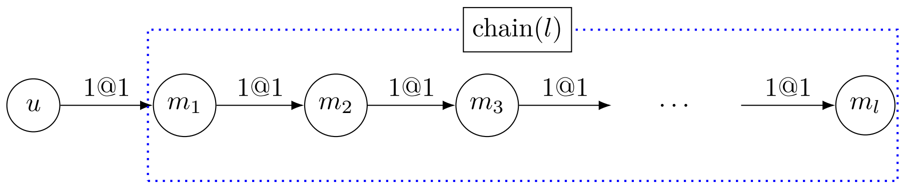


Figure 2.15: The chain gadget. In any valid schedule, either $p^1_{u,m_1} < 1$ and all vertices $m_i$ are bankrupt, or $p^1_{u,m_1} = 1$ and no vertices $m_i$ are bankrupt.

We now define variable nodes $\{s_i : 1 \leq i \leq n\}$, literal nodes $\{x_i, \neg x_i : 1 \leq i \leq n\}$, clause nodes $\{q_j : 1 \leq j \leq m\}$ and a sink node $d$ analogously to the proof of Theorem 2.7 and include the debts as depicted in Fig. 2.16 so as to obtain our IDM instance $(G, D, A^0)$. Note that the chain gadgets corresponding to the different literal nodes are all distinct and $|c_j|$ denotes the number of literals in the clause $c_j$ of $\phi$.

Note that because all debts of $(G, D, A^0)$ are due at an exact time, rather than over an interval, reasoning in the AoN variant is identical to reasoning in the PP variant.

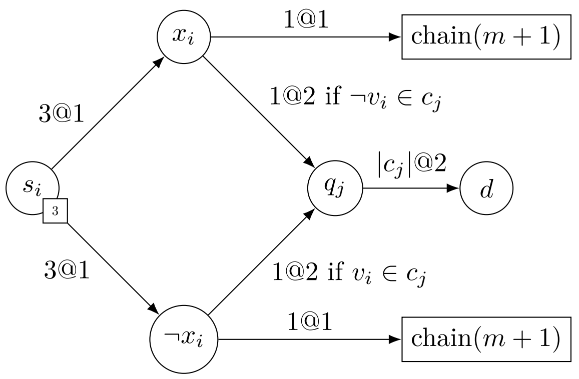


Figure 2.16: An IDM instance illustrating the reduction from 3-SAT 3 to Bankruptcy Maximization, using chain gadgets.

**Claim 2.13.1.** In any valid schedule $\sigma$ for $(G, D, A^0)$ in which $c \geq 0$ variable nodes $s_i$ either pay €3 to $x_i$ or €3 to $\neg x_i$:

- all $n$ variable nodes are bankrupt
- exactly $n$ literal nodes are bankrupt
- exactly $c(m+1)$ chain nodes are bankrupt.

Consequently, this amounts to exactly $2n + c(m+1)$ bankrupt nodes with any other bankrupt nodes necessarily being clause nodes.

*Proof.* Suppose that in the valid schedule $\sigma$, $s_i$, for some $1 \leq i \leq n$, pays €1 to a node from $\{x_i, \neg x_i\}$ at time 1 and €2 to the other node from $\{x_i, \neg x_i\}$ at time 1. Since both $x_i$ and $\neg x_i$ have €1 at time 1, both must pay €1 to their corresponding chain gadget; so, none of the nodes of either of these chain gadgets is bankrupt. As any variable and its negation both appear at least once in some clause of $\phi$, exactly one of the nodes $x_i$ and $\neg x_i$ is bankrupt at time 2.

Alternatively, suppose that $s_j$, for $1 \leq j \leq n$, pays €3 to either $x_j$ or $\neg x_j$ at time 1. So, the literal node to which $s_j$ makes no payment is bankrupt at time 1 as are all the nodes of its corresponding chain gadget.

In any case, we have: $n$ variable nodes that are bankrupt; $n$ literal nodes that are bankrupt; and $c(m+1)$ chain nodes that are bankrupt. This results in $2n + c(m+1)$ bankrupt nodes. As the sink node $d$ cannot be bankrupt, the claim follows. □

**Claim 2.13.2.** If $((G, D, A^0), 2n + n(m+1) + m)$ is a yes-instance of Bankruptcy Maximization then $\phi$ is a yes-instance of 3-Sat-3.

*Proof.* Suppose that there exists a valid schedule $\sigma$ that results in at least $2n+n(m+1)+m$ bankruptcies. So, by Claim 2.13.1, every variable node $s_i$ pays €3 to either $x_i$ or $\neg x_i$ and also every clause node $q_j$ is bankrupt in $\sigma$. The reason a clause node $q_j$ is bankrupt is because there is some literal $v_i$ or $\neg v_i$ in clause $c_j$ but where $\neg x_i$ or $x_i$, respectively, receives no payment from $s_i$. Define the truth assignment $X$ on the variables of $\phi$ so that $X(v_i) = \mathit{True}$ iff $x_i$ receives a payment of €3 from $s_i$, for $1 \leq i \leq n$. This truth assignment satisfies every clause of $\phi$. □

**Claim 2.13.3.** If $\phi$ is a yes-instance of 3-Sat-3 then $((G, D, A^0), 2n+n(m+1)+m)$ is a yes-instance of Bankruptcy Maximization.

*Proof.* Suppose that there is a satisfying truth assignment $X$ for $\phi$. Consider the following schedule $\sigma$:

- at time 1, every $s_i$ pays: €3 to $x_i$ if $X(v_i) = True$; and €3 to $\neg x_i$ if $X(v_i) = False$
- if $x_i$ (resp. $\neg x_i$) received €3 from $s_i$ at time 1 then:
  - at time 1, it pays €1 to its corresponding chain gadget so as to satisfy all debts in the gadget
  - at time 2, it pays €1 to each clause node $q_j$ for which the literal $\neg v_i \in c_j$ (resp. $v_i \in c_j$)
- if $x_i$ (resp. $\neg x_i$) received no payment from $s_i$ at time 1 then at times 1 and 2 it can make no payments
- each $q_j$ makes a payment of however many euros it has to $d$ at time 2 (note that it never received more than €$|c_j|$).

The schedule $\sigma$ is clearly valid. By Claim 2.13.1, we have at least $2n + n(m + 1)$ bankrupt nodes with any additional bankrupt nodes necessarily clause nodes. Consider some clause node $q_j$ containing some literal $v_i$ so that $X(v_i) = True$. By definition, $\neg x_i$ receives no payment from $s_i$ at time 1 and so the debt of €1 at time 2 from $\neg x_i$ to $q_j$ is not paid. Consequently, $q_j$ is bankrupt. Hence, we have exactly $2n + n(m + 1) + m$ bankrupt nodes and the claim follows. □

Note that the above reasoning clearly holds in the AoN variant (since in the schedules we consider all debts are paid either in full or not at all) as well as in the FP variant (since in order for $s_i$ to bankrupt one of the chains attached to $x_i$ and $\neg x_i$ it must pay *strictly* less than £1 to the bankrupt node and hence *at least* £2 to the “surviving” node). As the construction of $((G, D, A^0), 2n+n(m+1)+m)$ can be completed in time polynomial in $n$, the result follows. □

Just as we did with Perfect Scheduling in Theorem 2.12, we can restrict Bankruptcy Maximization in the AoN variant so that any IDM $(G, D, A^0)$ in any instance is such that $G$ is a directed path of bounded length (here 2).

**Theorem 2.14.** Consider the problem Bankruptcy Maximization restricted so that every instance $((G, D, A^0), k)$ is such that $G$ is a directed path of length 3. If, further, $T$ is restricted to be 2 then the resulting problem is weakly NP-complete in the AoN variant.

*Proof.* We reduce from the weakly NP-complete problem Subset Sum defined as follows (see [71]).

<u>Subset Sum</u>

Instance: a multi-set of integers $S = \{a_1, a_2, \ldots, a_n\}$ and an integer $k$

Yes-instance: there exists a subset $S_1$ of $S$ so that the sum of the numbers in $S_1$ is $k$.

In general, an instance $S = \{a_1, a_2, \ldots, a_n\}$ has size $n\beta$ where $\beta$ is the least number of bits required to express any of the integers of $S$ in binary. Let $S$ be an instance of Subset Sum of size $n\beta$. By doubling all integers if necessary, we may assume that every integer of $S$ is at least 2. Consider the IDM instance $(G, D, A^0)$ in Fig. 2.17 (for which $T = 2$). The value $A$ in Fig. 2.17 is the sum of all integers in $S$. Note that the time taken to construct $(G, D, A^0)$ from $S$ is polynomial in $n\beta$.

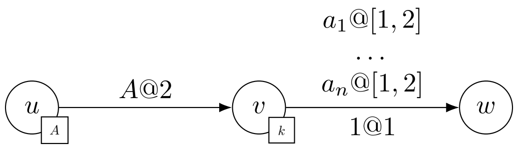


Figure 2.17: An IDM instance corresponding to an instance of Subset Sum.

**Claim 2.14.1.** If $((G, D, A^0), 1)$ is a yes-instance of Bankruptcy Maximization then $S$ is a yes-instance of Subset Sum.

*Proof.* Suppose that $\sigma$ is a valid schedule within which there is at least 1 bankruptcy in the IDM instance $(G, D, A^0)$. The nodes $u$ and $w$ are never bankrupt; so, $v$ must be bankrupt within $\sigma$. At time 2, the node $v$ necessarily pays off all of the unpaid debts to $w$ of monetary amount greater than €1, as it receives sufficient funds from $u$ at time 2 to do this. Hence, $v$ must be bankrupt at time 1; that is, $v$ does not pay its debt to $w$ of monetary amount €1 at time 1. As $\sigma$ is valid, $v$ is not withholding at time 1 and the only way for this to happen is for $v$ to pay debts to $w$ amounting to €$k$. That is, we have a subset of integers of $S$ whose total sum is $k$; that is, $S$ is a yes-instance of Subset Sum. The claim follows. $\square$

**Claim 2.14.2.** If $S$ is a yes-instance of Subset Sum then $((G, D, A^0), 1)$ is a yes-instance of Bankruptcy Maximization.

*Proof.* Suppose that the subset $S_1$ of $S$ is such that $sum(S_1) = k$. W.l.o.g. let $S_1 = \{a_1, a_2, \ldots, a_r\}$. Define the schedule $\sigma$ as: at time 1, $v$ pays the debts $a_1@[1,2], \ldots, a_r@[1,2]$; and at time 2, $u$ pays its debt to $v$ and $v$ pays the debts $a_{r+1}@[1,2], \ldots, a_n@[1,2]$, and the overdue debt $1@1$. Note that this is a valid schedule, as no node is withholding at any time, within which $v$ is bankrupt. The claim follows. $\square$

The main result follows. $\square$

### 2.3.4 Polynomial-time algorithms

In this section we show that Bailout Minimization in the FP variant is solvable in polynomial-time and also that Bailout Minimization in the PP variant is solvable in polynomial-time when our IDM instances are restricted to out-trees. We begin with the FP variant result.

**Theorem 2.15.** The problem Bailout Minimization in the FP variant is solvable in polynomial time.

*Proof.* A solution to FP Bailout Minimization is a bailout vector $B$ of size $|V|$ together with a schedule $\sigma$ consisting of $|E|T$ payment values $p_e^t$. We describe below how an instance $((G, D, A^0), b)$ of Bailout Minimization can be encoded as a linear program (LP), which can then be solved in polynomial time.

Our variables are:

- Bailout variables $\{B[v] | v \in V\}$ (altogether $|V|$ variables),
- Payment variables $\{p_e^t | e \in e, t \in [T]\}$ (altogether $|E|T$ variables), and
- Income variables $\{I_v^t | v \in V, t \in [T]\}$, outgoing variables $\{O_v^t | v \in V, t \in [T]\}$, and cash asset variables $\{c_v^t | v \in V, t \in [0, T]\}$ (altogether $3 \cdot |V| \cdot T + |V|$ variables).

In the below, for $a, b \in \mathbb{N}_0$, $[a, b]$ denotes the set $\{a, a+1, \ldots, b\}$, and we write $[b]$ as shorthand for $[1, b]$. Our constraints are:

- The total bailout is at most $b$:

$$\sum_v B[v] \leq b$$

- The starting cash assets of a node (at time 0) are its external assets (specified by $A^0$) plus any bailout it receives. For each $v \in V$:

$$c_v^0 = A^0[v] + B[v]$$

- No debt is paid early, and all payments are non-negative. For each $e \in E$ and $t \in [0, T]$:

$$p_e^t \begin{cases} = 0, & \text{if } t < D_{t_1}(e) \\ \geq 0, & \text{otherwise} \end{cases}$$

- The income (resp. outgoings) of a node at some time are obtained by summing over payments into (resp. out of) that node at each time. These then can be used to compute external (cash) assets at all nodes and times. For each $v \in V$ and $t \in [T]$:

$$I_v^t = \sum_{e \in E_{in}(v)} p_e^t \quad (\text{resp. } \; O_v^t = \sum_{e \in E_{out}(v)} p_e^t)$$

$$c_v^t = c_v^{t-1} + I_v^t - O_v^t$$

- No bank has negative assets at any point. For each $v \in V, t \in [T]$:

$$c_v^t \geq 0$$

- Each debt is paid in full within its interval. This guarantees that there are no bankruptcies in the schedule (and that no banks are withholding, a validity constraint). For each $e \in E$ with $D(e) = (a, t_1, t_2)$:

$$\sum_{t \in [t_1, t_2]} p_e^t = a$$

Recall from our discussion of canonical instances in Section 2.4 that we may assume $T$ is at most $2|E|$. Then we have $O(nm + m^2)$ variables and $O(nm + m^2)$ constraints. If the largest integer in the input instance $((G, D, A^0), b)$ required $\beta$ bits to encode, then our constructed LP has size polynomial in $n + m + \beta$. Any assignment to $B$ and to the payment variables $p_e^t$ satisfying the above is necessarily a perfect valid schedule for $((G, D, A^0), b)$. As linear programs can be solved in polynomial-time, our result follows. $\square$

Note the limitations of the use of linear programming for other problems. For Bailout Minimization in the PP variant, proceeding as in the proof of Theorem 2.15 results is an integer linear program, the solution of which is NP-complete in general. Moreover, we have already proven Perfect Scheduling, the special case of Bailout Minimization with $b$ fixed to 0, to be NP-complete in the PP variant through the proofs in Theorems 2.9, 2.10 and 2.11. As regards trying to use linear programming for Bankruptcy Minimization in the FP variant, it is not possible to express a constraint on the number of bankruptcies through a linear combination of the payment variables; indeed, we have already proven Bankruptcy Minimization in the FP variant to be NP-complete in Theorems 2.7 and 2.8.

For the AoN and PP variants, by restricting the temporal properties of the IDM instances considered, we obtain tractability of Bailout Minimization, namely when all debts are due at an exact time.

**Theorem 2.16.** The problem (AoN/PP/FP) Bailout Minimization is solvable in polynomial time when restricted to inputs $(G, D, A^0)$ such that $D_{t_1} = D_{t_2}$.

*Proof.* Let $(G, D, A^0)$ be an IDM instance satisfying the above. In such an instance, all debts are due at an exact point in time, rather than an interval. For convenience, we use $D_t$ as shorthand for either of $D_{t_1}$ or $D_{t_2}$. By definition, for any bailout vector $B$ (including the all-zero vector) a perfect schedule for $(G, D, A^0 + B)$ is one in which every debt is paid in full and on time. Let $\sigma$ be the schedule defined by $p_e^{D_t(e)} = D_a(e)$ for each edge $e$, with all other payment variables equal to zero. Clearly, for any vector $B$, a perfect schedule for $(G, D, A^0 + B)$ exists if and only if $\sigma$ is a valid schedule (and hence a perfect schedule).

Moreover, we can efficiently compute a vector $B$ of minimum sum such that $\sigma$ is a perfect schedule for $(G, D, A^0 + B)$. For each vertex $v$ and time $t$, compute $c_v^t$ under $\sigma$ for the instance $(G, D, A^0)$. Note that if $(G, D, A^0)$ does not admit a perfect schedule then $c_v^t$ will be negative for some $v$ and $t$, and $\sigma$ is not a valid schedule for that instance (without a bailout). Denote the minimum (again, possibly negative) cash assets of $v$ at any time by $c_v^{\min}$. Compute $b_v := \max(-1 \cdot c_v^{\min}, 0)$ for each $v$, and let $B = (b_v | v \in V)$. By construction, $\sigma$ is a perfect schedule for $(G, D, A^0 + B)$, and $\sigma$ is not a perfect schedule for $(G, D, A^0 + B')$ for any $B'$ with $\text{sum}(B') < \text{sum}(B)$.

All of our arguments hold in all three variants (AoN, PP, and FP), and the result follows. □

Interestingly, Theorem 2.16 is the only positive result we derive for the All-or-Nothing setting. We also obtain tractability results for the problem PP Bailout Minimization if we restrict the structure of IDM instances.

**Theorem 2.17.** The problem Bailout Minimization in the PP variant is solvable in polynomial-time when our IDM instances are restricted to out-trees.

*Proof.* Let $((G, D, A^0), b)$ be an instance of Bailout Minimization so that $G$ is an out-tree. Suppose that $G$ has node set $\{u_i : 1 \leq i \leq n\}$. We need to decide whether we can increase the initial external assets of each node $u_i$ by $b_i$ so that $\sum_i b_i \leq b$ and $(G, D, A^0 + B)$ has a perfect schedule, where $B = (b_1, b_2, \ldots, b_n)$; that is, whether $(G, D, A^0)$ is '$b$-bailoutable' via a *$b$-bailout vector* $B$. Our intention is to repeatedly amend $(G, D, A^0)$ so that $(G, D, A^0)$ is '$b$-bailoutable' iff the resulting IDM instance is '$b'$-bailoutable', for some amended $b'$; in such a case, we say that the two problem instances are *equivalent*. We will then work with the (simplified) amended instance.

We proceed as follows. First, identify nodes $v$ for which, at any time $t$, the sum of all debts $v$ must pay by time $t$ minus the sum of all debts which could be paid to $v$ by time $t$ exceeds $v$'s initial external assets $c_v^0$. We call these nodes *prefix-insolvent*. Note that if a node $v$ is prefix-insolvent then under any perfect schedule $\sigma$, we would have $I_v^{[t]} + c_v^0 < O_v^{[t]}$, violating a validity constraint, and hence there is no such perfect schedule. Also note that every insolvent node is prefix-insolvent (namely by taking $t = T$). For any node that is prefix-insolvent, increase the initial external assets by the minimal amount that causes the node to cease to be prefix-insolvent and simultaneously decrease the bailout amount by this value. Our new instance is clearly equivalent with our initial instance. If, in doing this, the bailout amount becomes less than 0 then we answer 'no' and we are done. So, we may assume that none of our nodes is prefix-insolvent.

Before we start, we make a simple amendment to the debts $D$: we replace any debt from node $u$ to node $v$ of the form $a@[t_1, t_2]$, where $a > 1$, with $a$ distinct debts $1@[t_1, t_2]$. Our resulting instance, with bailout $b$, is equivalent to our initial instance, with bailout $b$, as we are working within the PP variant. This amendment simplifies some of the reasoning coming up. Note that it may be necessary to simulate this operation rather than actually performing it (since if $a$ is exponential in the instance size then the operation takes exponential time), but that the reasoning which follows can easily be "scaled up" to deal with non-unit amounts.

Consider a leaf node $v$ and its parent $u$ in the out-tree $G$. Replace every debt of the form $1@[t_1, t_2]$ from $u$ to $v$ by the debt $1@t_2$ and denote the revised instance by $(G, D', A^0)$. Let $\sigma$ be a perfect valid schedule for $(G, D, A^0+B)$ (here, and throughout, we write $B$ to denote some $b$-bailout vector; that is, some assignment of resource to the nodes of $G$ so that the total bailout amount does not exceed the total available $b$). Define the schedule $\sigma'$ for $(G, D', A^0 + B)$ by amending any payment from $u$ to $v$ of some debt $1@[t_1, t_2]$ so that the payment is made at time $t_2$. The schedule $\sigma'$ is clearly a perfect valid schedule for $(G, D', A^0 + B)$. Furthermore, any perfect valid schedule for $(G, D', A^0 + B)$ is a perfect valid schedule for $(G, D, A^0 + B)$. Hence, we can replace $((G, D, A^0), b)$ by $((G, D', A^0), b)$, as these instances are equivalent. We can proceed as above for every leaf node and its parent and so assume that all debts from a parent to a leaf are due at some specific time only; that is, have a singular time-stamp and are of the form $1@t$. Note also that no node of $G$ is insolvent.

Suppose that we have two leaf nodes $v$ and $w$ with the same parent $u$. We can replace $v$ and $w$ with a 'merged' node $vw$ so that all debts from $u$ to $v$ or from $u$ to $w$ are now from $u$ to $vw$. Our initial problem instance is clearly equivalent to our amended problem instance (note that we never assign bailout resource to either $v$ or $w$ as this is pointless). We can proceed likewise for all such triples $(u, v, w)$. Hence, we may assume that our digraph $G$ is such that: no two leaves have a common parent; all debts to a leaf node have monetary amount €1 and have a singular time-stamp; and no node in $G$ is prefix-insolvent.

Suppose that we have a leaf node $w$ that is the only child of its parent node $v$ whose parent is $u$ (such a node $w$ exists unless our tree is a star – an easy case we deal with at the end of the proof). We may assume that $v$ has no initial external assets as we would simply use these assets to pay as many debts to $w$ as possible (in increasing order of time-stamp); that is, we could remove all these debts from $D$ along with the corresponding amount from the initial external assets of $v$. If the result of doing this is that there are no debts from $v$ to $w$ then we remove $w$ from $G$ and any remaining initial external assets from $v$. We would then repeat all of the above amendments until it is the case that our nodes $u$, $v$ and $w$ are such that $v$ has no initial external assets.

By the above, every debt from $v$ to $w$ is of the form $1@t$. Let $t'$ be the minimum time-stamp for all debts from $v$ to $w$ and let $d_v$ be a debt from $v$ to $w$ of the form $1@t'$. Consider the debts from $u$ to $v$: these have the form $1@[t_1, t_2]$. There are various cases:

$(a)$ there is a debt $d_u$ from $u$ to $v$ of the form $1@[t_1, t_2]$ where $t_2 \leq t'$

$(b)$ there is no debt from $u$ to $v$ of the form $1@[t_1, t_2]$ where $t_2 \leq t'$ but there is a debt from $u$ to $v$ of the form $1@[t_1, t_2]$ where $t_1 \leq t' \leq t_2$

$(c)$ there is no debt from $u$ to $v$ of the form $1@[t_1, t_2]$ where $t_1 \leq t'$.

The nodes $u$, $v$ and $w$ can be visualized as in Fig. 2.18(d) and the three cases above in Fig. 2.18(a)–(c).

Case $(a)$: We amend $G$ by: introducing a new node $v'$ whose parent is $u$ and a new debt $d'$ from $u$ to $v'$ of the form $1@[t_1, t_2]$; removing the debt $d_u$ from $u$ to $v$; and removing the debt $d_v$ from $v$ to $w$. Denote this revised IDM instance by $(G', D', A^0)$.

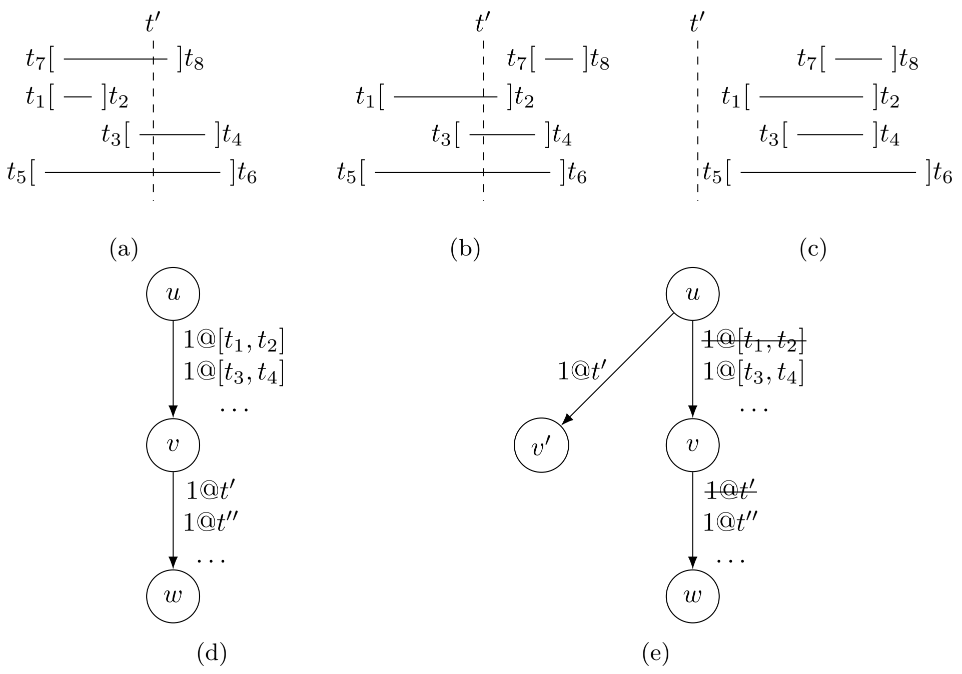


Figure 2.18: Cases when a leaf has a parent with one child in our algorithm for PP Bailout Minimization on out-trees.

Suppose that there exists a perfect valid schedule $\sigma$ for $(G, D, A^0 + B)$. Define the schedule $\sigma'$ for $(G', D', A^0 + B)$ from $\sigma$ by: changing the payment from $u$ to $v$, at time $t$ where $t_1 \leq t \leq t_2 \leq t'$ and covering the debt $d_u$, so that the payment is made from $u$ to $v'$ at time $t$ (so as to cover the new debt $d'$); and dropping the payment, at time $t'$, that covers the debt $d_v$. The resulting schedule $\sigma'$ is clearly valid and perfect. Conversely, suppose that we have a perfect valid schedule $\sigma'$ for $(G', D', A^0 + B)$. Define the schedule $\sigma$ for $(G, D, A^0 + B)$ from $\sigma'$ by: changing the payment from $u$ to $v'$ at time $t$ where $t_1 \leq t \leq t_2 \leq t'$ and covering the debt $d'$, so that the payment is made from $u$ to $v$ at time $t$, so as to cover the debt $d_u$; and using the €1 received by $v$ so as to cover the debt $d_v$ from $v$ to $w$. The resulting schedule $\sigma$ is clearly a perfect valid schedule for $(G, D, A^0 + B)$. Consequently, $((G, D, A^0), b)$ and $((G', D', A^0), b)$ are equivalent and we can work with $((G', D', A^0), b)$.

Case $(b)$: Let $D_u$ be the set of debts from $u$ to $v$ and let $D_v$ be the set of debts from $v$ to $w$. Order the $k$ debts of $D_v$ in increasing order of time-stamp as $d_v = d_1, d_2, \ldots, d_k$ where the corresponding time-stamps are $t' = \bar{t}_1, \bar{t}_2, \ldots, \bar{t}_k$ (there may be repetitions). Suppose that for some $1 \leq i \leq k$, the number of debts in $D_u$ of the form $1@[t_1, t_2]$ with $t_1 \leq \bar{t}_i$ is strictly less than $i$. Consequently, at time $\bar{t}_i$, the debt $d_i$ cannot be paid and we necessarily need to give $v$ some bailout amount, €$c \geq 1$ say, to cover the $c$ debts that cannot be paid by $v$ at time $\bar{t}_i$. We do this and reduce the overall bailout amount by €$c$. We then delete debts $d_1, d_2, \ldots, d_c$ from $G$ and remove the bailout amount of €$c$ from $v$. If doing this results in there being no remaining debts from $v$ to $w$ then we delete $w$ from $G$. Irrespective of this, the resulting instance $((G', D', A^0), b')$, where $b' = b - c$, is equivalent to $((G, D, A^0), b)$. We would then repeat all of the amendments above and so w.l.o.g. we may assume that we are in Case $(b)$ and for every $1 \leq i \leq k$, the number of debts in $D_u$ of the form $1@[t_1, t_2]$

with $t_1 \leq \bar{t_i}$ is at least $i$. In particular, there are at least $k$ debts in $N_u$.

Suppose that $\sigma$ is a valid perfect schedule for $(G, D, A^0 + B)$, for some $B$ where $v$ receives a bailout amount of €$c > 0$. Suppose further that there is no $B'$ where there is a valid perfect schedule for $(G, D, A^0 + B')$ with $v$ receiving a bailout of less than €$c$. The reason that $v$ receives the bailout amount of €$c$ is that in the schedule $\sigma$, if we ignore the payments by $v$ that use the bailout amount at $v$ then there are $c$ debts from $d_1, d_2, \ldots, d_k$ that are not paid on time; let us call these debts the 'bad' debts. Note that for each bad debt, there is a debt from $u$ to $v$ that *might* have been paid at a time early enough to cover the debt but wasn't. Let $e_1, e_2, \ldots, e_c$ be distinct debts from $D_u$ that might have been paid earlier so as to enable the payment of the bad debts. Amend the bailout $B$ so that the €$c$ formerly given as bailout to $v$ is now given to $u$ and denote this revised bailout by $B'$. Revise the schedule $\sigma$ so that for each $1 \leq i \leq c$, €1 of bailout at $u$ is used to pay the debt $e_i$ at the earliest time possible. Doing so results in us being able to pay all bad debts on time; hence, we have a perfect schedule for $(G, D, A^0 + B')$ where $v$ receives no bailout. This yields a contradiction and so if there is a bailout $B$ and a valid perfect schedule for $(G, D, A^0 + B)$ then there is a bailout $B'$ and a valid perfect schedule for $(G, D, A^0 + B')$ where $v$ receives no bailout funds. We shall return to this comment in a moment.

From the debts of $D_u$, we choose a debt $d_u = 1@[t_1, t_2]$, where $t_1 \leq t' \leq t_2$, so that from amongst all of the debts of $D_u$ of the form $1@[t_3, t_4]$, where $t_3 \leq t' \leq t_4$, we have that $t_2 \leq t_4$; that is, from all of the debts of $D_u$ that 'straddle' $t'$, $d_u$ is a debt whose right-most time-stamp is smallest. We amend $G$ by: introducing a new node $v'$ and a new debt $d'$ from $u$ to $v'$ of the form $1@t'$; removing the debt $d_u$ from $u$ to $v$; and removing the debt $d_v$ from $v$ to $w$. Denote this revised IDM instance by $(G', D', A^0)$; it can be visualized as in Fig. 2.18(e).

Suppose that $\sigma$ is a valid perfect schedule for $(G, D, A^0 + B)$, for some $B$. From above, we may assume that there is no bailout to node $v$ in $B$. Consider the payment by $v$ to $w$ of the debt $d_v$. If the actual €1 that pays this debt came from the payment of a debt from $D_u \setminus \{d_u\}$ of the form $1@[t_3, t_4]$ (where $t_3 \leq t' \leq t_4$), then we can amend $\sigma$ so that we use this €1 to pay the debt $d_u$ at the time $t'$ and use the €1 that paid the debt $d_u$ to pay the debt $1@[t_3, t_4]$ (at whatever time $d_u$ was paid – without loss in the interval $[t' + 1, t_2] \subset [t' + 1, t_4]$, since $t'$ is the earliest time any debt to $v$ is due); that is, we swap the times of the payment of the debts $d_u$ and $1@[t_3, t_4]$ in $\sigma$ except that we now pay $d_u$ at time $t'$. If the actual €1 that pays $d_v$ came from the payment of $d_u$ then we can amend the payment time of $d_u$ to $t'$ (if necessary).

Build a schedule $\sigma'$ in $(G', D', A^0 + B)$ from $\sigma$ by: instead of paying $d_u$ (at time $t'$), we pay the new debt $d'$ from $u$ to $v'$; and we remove the payment of the debt $d_v$. The schedule $\sigma'$ is clearly a valid perfect schedule of $(G', D', A^0 + B)$. Conversely, if $\sigma'$ is a valid perfect schedule of $(G', D', A^0 + B)$, we can build a schedule $\sigma$ for $(G, D, A^0 + B)$ from $\sigma'$ by: instead of paying the debt $d'$ (at time $t'$), we pay the debt $d_u$ at time $t'$; and we use this €1 to pay immediately the debt $d_u$. The schedule $\sigma$ is clearly a valid perfect schedule of $(G, D, A^0 + B)$. Hence, $((G, D, A^0), b)$ and $((G', D', A^0), b)$

are equivalent and we can work with $((G', D', A^0), b)$.

Case $(c)$: Suppose that there is no debt from $u$ to $v$ of the form $1@[t_1, t_2]$ where $t_1 \leq t'$. This case cannot happen as we have ensured that no node of $G$ is insolvent.

By iteratively applying all of the amendments to the instance $((G, D, A^0), b)$, as laid out above, we reduce $((G, D, A^0), b)$ to an equivalent instance $((G', D', (A')^0), b')$ where $G$ consists of a solitary directed edge and all debts have a singular time-stamp. The process of reduction can clearly be undertaken in time polynomial in the size of the initial instance $((G, D, A^0), b)$ and the resulting instance $((G', D', (A')^0), b')$ can clearly be solved in time polynomial in the size of the initial instance $((G, D, A^0), b)$. Hence, Bailout Minimization is solvable in polynomial-time. □

Our polynomial-time algorithm for Bailout Minimization, and so Perfect Scheduling, in the PP variant in Theorem 2.17 when we restrict to out-trees contrasts with the NP-completeness of Perfect Scheduling when we restrict to directed acyclic graphs or multiditrees, as proven in Theorem 2.9 and Theorem 2.10, respectively.

## 2.4 Conclusion and open problems

This chapter introduces the *Interval Debt Model (IDM)*, a new model seeking to capture the temporal aspects of debts in financial networks. We investigate the computational complexity of various problems involving debt scheduling, bankruptcy and bailout with different payment options (All-or-nothing (AoN), Partial (PP), Fractional (FP)) in this setting. We prove that many variants are hard even on very restricted inputs but certain special cases are tractable. For example, we present a polynomial time algorithm for PP Bailout Minimization where the IDM graph is an out-tree. However, for a number of other classes (DAGs, multitrees, total assets are €1), we show that the problem remains NP-hard. This leaves open the intriguing question of the complexity status of problems which are combinations of two or more of these constraints, most naturally on multitrees which are also DAGs, an immediate superclass of our known tractable case.

An interesting result of ours is the (weak) NP-completeness of Bankruptcy Minimization on a fixed, 32-node footprint graph (with edge multiplicity unbounded) in Theorem 2.8. It is noteworthy that constantly many nodes suffice to express the complexity of any problem in NP, and this leads to several open questions. Does the same hold when integers must be encoded in unary? We know this is true for the AoN case (as shown in Theorem 2.12). What is the smallest number $n$ such that a family of $n$-node (FP/PP) Bankruptcy Minimization instances is NP-complete? From the other side, what is the largest number $n$ such that any $n$-node (FP/PP) Bankruptcy Minimization instance may be solved in polynomial time, and with what techniques?

We prove that FP Bailout Minimization is polynomial-time solvable by expressing it as a Linear Program. Can a similar argument be applied to some restricted version of FP Bankruptcy Minimization (which is NP-Complete, in general)? A natural generalization is simultaneous Bailout and Bankruptcy minimization i.e. can we allocate €$b$ in bailouts such that a schedule with at most $k$ bankruptcies becomes possible. Variations of this would be of practical interest. For example, if regulatory authorities can allocate bailouts as they see fit, but not impose specific payment times, it would be useful to consider the problem of allocation of €$b$ in bailouts such that the maximum number of bankruptcies in any valid schedule is at most $k$. Conversely, where financial authorities can impose specific payment times, the combination of the problems Bankruptcy Minimization and Bailout Minimization would be more applicable.

Finally, can we make our models more realistic and practical? How well do our approaches perform on real-world financial networks? Can we identify topological and other properties of financial networks that may be leveraged in designing improved algorithms? What hardness or tractability results hold for variants in which the objective is, instead of the number of bankruptcies, the total amount of unpaid debt (or any other objective, for that matter)?

Chapter 3

# Temporal Reachability Dominating Sets: contagion in temporal graphs

*This chapter is based on joint work with Laura Larios-Jones, a PhD student at the University of Glasgow. Sections 3.2.2 and 3.3.2 are solely attributed to the author, whereas Sections 3.2.4, 3.4.3 and 3.5.1 are solely attributed to Larios-Jones (these are included here because of the results' use in framing the other contributions in this chapter). All other results in this chapter were written and researched collaboratively.*

## 3.1 Introduction

A natural problem in the study of networks is that of finding a small set of individuals which together can affect the entire network. This problem is practically relevant in identifying influential entities in social networks, in gauging the risk of viral spread in biological networks, or in choosing sources to broadcast from in wireless networks. We study this problem through the lens of *temporal graphs*. That is, graphs which change over time, capturing the dynamic nature intrinsic to many real-world networks.

We formalize the notion of a set of sources which together reach the whole temporal graph as a *Temporal Reachability Dominating Set*, or TaRDiS. We study the complexity of the problem of finding a TaRDiS of a given size in a given temporal graph. Later, we ask: if we can choose when connections between individuals exist, can we maximize the size of the minimum TaRDiS? That is, for viral spread, can we guarantee that the entire population will not be contaminated by at most $k$ initial infections. We refer to this problem as MaxMinTaRDiS.

Our problems sit at the intersection of reachability and covering, which are two rich classes of (temporal) graph problems. Reachability is a fundamental concept in temporal graph theory. Temporal reachability and related problems have been the subject of extensive study since the seminal work in the field by Kempe, Kleinberg and Kumar [66]. Several works focus on the complexity of choosing or modifying times to optimize reachability [66, 72, 23, 73, 74], while some consider as input temporal graphs in which times are immutably fixed [18, 75, 76, 77, 78]. A substantial and closely related stream of work to ours is the study of *temporal connectivity* – the property that any single vertex is, alone, a TaRDiS [74, 18, 17, 79, 80, 66, 26, 81]. In [18], *pivotability* is introduced to construct some extreme classes of connected temporal graph. A *pivot* vertex $p$ in a temporal graph is a vertex reachable from every vertex in the graph by some time $t$ and which reaches every vertex in the graph by a temporal path departing after time $t$. Thus having a pivot is a strictly stronger property than temporal connectivity. A comparison between our problems and some others in the literature is shown in Table 3.1.

MaxMinTaRDiS is closely related to the MinMaxReach problem studied by Enright, Meeks and Skerman [23], in which the objective is to minimize the maximum number of individuals reached by any single vertex. TaRDiS, on the other hand, vastly generalizes Casteigts's [82] notion of $\mathcal{J}^{1\forall}$ connectivity, in which the question is whether any single vertex reaches the entire network. This framing of reachability asks what the worst-case spread is from a single source in the temporal graph. In reality, studied populations are often infected by several individuals. For example, SARS-CoV-2 had been independently introduced to the UK at least 1300 times by June 2020 [83]. This, a dynamic population infected by many sources, is precisely the setting motivating our TaRDiS problem.

Our work is focused on the computational (parameterized) complexity of TaRDiS and MaxMinTaRDiS, which depends heavily on which formalisation of the problems is considered. In particular, instantaneous spread through a large swath of the population, while realistic in some computer networks, is inconsistent with viral spread in a biological system. Further, should multiple interactions between the same pair of individuals be allowed? Lastly, should it be possible for a single individual to simultaneously interact with several others?

We consider all combinations of answers to these questions, and in all cases for both problems: show that the problem is computationally hard in general; identify the maximum lifetime (number of discrete times which may appear in the input) such that the problem remains tractable; and provide (parameterized) algorithms. Interestingly, we also show through a nontrivial proof that MaxMinTaRDiS generalizes the well-studied Distance-3 Independent Set problem.

### 3.1.1 Problem Setting

We begin with some standard definitions. We denote by $[i, j]$ the set $\{i, i+1, \dots, j\}$ and $[j]$ the set $[1, j]$. Let $G = (V, E)$ be an undirected graph with $(u, v) \in E$. We say that $u$ and $v$ are *adjacent* (also *neighbours*) and that the edge $(u, v)$ is *incident* to both $u$ and $v$. If $S \subseteq V$ is a set of vertices, we say an edge $(u, v)$ is incident to $S$ if one of its endpoints $u$ or $v$ is in $S$. For any vertex $v \in V$, the *closed neighbourhood* of $v$ is denoted $N[V] := \{v\} \cup \{u : u \text{ is adjacent to } v\}$. We say $G$ is *planar* if it can be drawn in a plane without any edges intersecting.

A *temporal graph* $\mathcal{G} = (V, E, \lambda)$ consists of a set of vertices $V$, a set of edges $E$ and a function $\lambda : E \to 2^{[\tau]} \setminus \{\emptyset\}$ which maps each edge to a discrete set of times where the *lifetime* $\tau \in \mathbb{N}$ of a temporal graph is the value of the latest timestep. We

| **Problem** | **Choose** | **such that** | **reach** | **Aim/ motivation** |
|---|---|---|---|---|
| MaxMinTaRDiS | times for edges | no coalition of $k$ nodes | all nodes | Minimize connectivity |
| TaRDiS | nothing | a coalition of $k$ nodes | all nodes | Assess connectivity |
| MinMaxReach [23] | times for edges | no single node | more than $k$ nodes | Minimize connectivity |
| $\mathcal{J}^{1\forall}$ connectedness [82] | nothing | a single node | all nodes | Assess connectivity |
| Reachability Inference [66] | times for edges | a designated root node | all "good" nodes and no "bad" nodes | Network design |
| ReachFast [84] | edges to delay | each source in the designated set | all nodes | Maximize connectivity |
| MinReachDelete [73, 22] | edges to delete | the specified coalition | at most $r$ nodes | Minimize connectivity |
| MinReachDelay [73] | edges to delay | the specified coalition | at most $r$ nodes | Minimize connectivity |
| Minimum Labelling [74] | times for edges | every node | all nodes | Maximise connectivity |
| TRLP [85] | time-edges to perturb | a single node | at least $k$ nodes | Maximise connectivity |
| Delay Better (Chapter 5b) | edges to delay | each specified start node | its designated target node(s) by some deadline | Maximise connectivity |

Table 3.1: Comparison of our problems TaRDiS and MaxMinTaRDiS to problems in the literature.

refer to $\lambda$ as the *temporal assignment* of $\mathcal{G}$. If $t \in \lambda(e)$ then we call the pair $(e, t)$ a *time-edge*, and say $e$ is *active* at time $t$. The set of all time-edges is denoted $\mathcal{E}$. We abuse notation and write $\lambda(u, v)$ to mean $\lambda((u, v))$.

For a static graph $G = (V, E)$, we denote the temporal graph $(V, E, \lambda)$ by $\mathcal{G} = (G, \lambda)$. We also use $V(\mathcal{G}), E(\mathcal{G})$ to refer to the vertex and edge sets of $\mathcal{G}$, respectively, and use $E_t(\mathcal{G})$ to refer to the set of edges active at time $t$, and call $G_t(\mathcal{G}) = (V(\mathcal{G}), E_t(\mathcal{G}))$ the *snapshot at time t*. When $\mathcal{G}$ is clear from the context we may omit it. Also, we use the convention that no snapshot is empty, which guarantees $\tau \leq |\mathcal{E}|$. The static graph $\mathcal{G}_{\downarrow} = (V, E)$ is referred to as the *footprint* of $\mathcal{G}$.

A *strict* (respectively *nonstrict*) *temporal path* from a vertex $u$ to a vertex $v$ is a sequence of time-edges $(e_1, t_1), \ldots, (e_l, t_l)$ such that $e_1 \ldots e_l$ is a static path from $u$ to $v$ and $t_i < t_{i+1}$ (resp. $t_i \leq t_{i+1}$) for $i \in [1, l-1]$. A vertex $u$ *temporally reaches* (or just "reaches") a vertex $v$ if there is a temporal path from $u$ to $v$. The *reachability set* $R_u(\mathcal{G})$ of a vertex $u$ is the set of vertices reachable from $u$. When the graph is clear from context, we may simply use $R_u$ to refer to the reachability set of a vertex

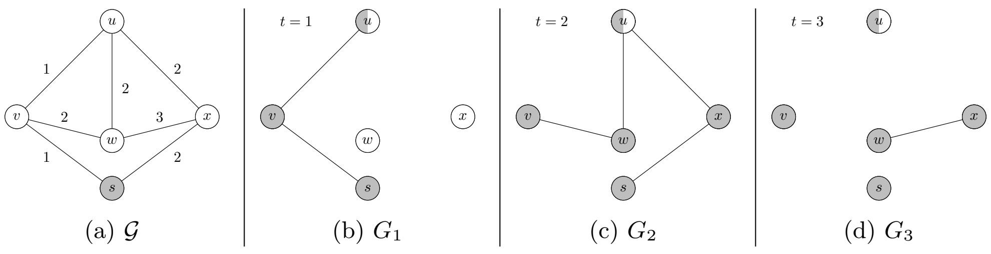


Figure 3.1: Spread in a temporal graph from source $s$ through snapshots. Vertices are shaded (half-shaded) when reached from $s$ by a strict (nonstrict) temporal path.

$u$. We say a vertex $u$ is reachable from a set $S$ if for some $v \in S$, $u \in R_v(\mathcal{G})$. A set of vertices $T$ is *temporal reachability dominated* by another set of vertices $S$ if every vertex in $T$ is reachable from $S$. Domination of and by single vertices is analogously defined. Strict and nonstrict reachability are illustrated in Figure 3.1. We differentiate between strict and nonstrict reachability by introducing a superscript $<$ or $\leq$ to the appropriate operators. For example, in Figure 3.1, $u$ is in $R_s^{\leq}$, but is not in $R_s^{<}$. Note that any strict temporal path is also a nonstrict temporal path, but the converse does not necessarily hold.

Casteigts, Corsini, and Sarkar [78] define three useful properties a temporal graph may exhibit. A temporal graph is called *simple* if each edge is active exactly once, *proper* if each snapshot has maximum degree one, and *happy* if it is both simple and proper. Figure 3.2 provides examples of the different types of temporal graph. For simple graphs, we define the temporal assignment as $\lambda : E \to [\tau]$ for convenience. We also use these three terms to describe the temporal assignment of a graph with the corresponding property. Happy temporal graphs have the property that any nonstrict temporal path is also a strict temporal path. Part of the utility of happy temporal graphs is that hardness results on them generalize to the strict and nonstrict settings. We can now introduce our protagonist.

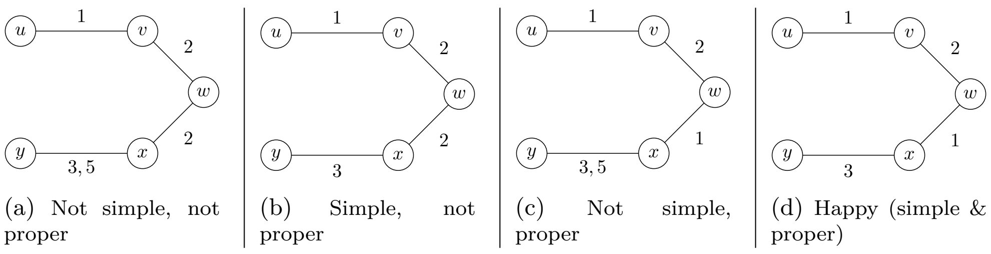


Figure 3.2: Four small examples of (non)simple and (non)proper temporal graphs.

**Definition 3.1** (TaRDiS)**.** In a temporal graph $\mathcal{G}$, a (strict/nonstrict) *Temporal Reachability Dominating Set* (TaRDiS) is a set of vertices $S$ such that every vertex $v \in V(\mathcal{G})$ is temporally reachable from a vertex in $S$ by a (strict/nonstrict) temporal path.

A minimum TaRDiS is a TaRDiS of fewest vertices in $\mathcal{G}$. We emphasize that, just as every strict temporal path is a nonstrict temporal path, every strict TaRDiS is a nonstrict TaRDiS. It is possible that the smallest nonstrict TaRDiS is strictly smaller than the smallest strict TaRDiS; for example, in Figure 3.2b, there is a nonstrict

TaRDiS of size 1 (namely $\{u\}$) but every strict TaRDiS has size at least 2. We now formally define our problems.

(Strict/Nonstrict) TaRDiS
*Input:* A temporal graph $\mathcal{G} = (V, E, \lambda)$ and an integer $k$.
*Question:* Does $\mathcal{G}$ admit a (strict/nonstrict) TaRDiS of size at most $k$?

The restriction to *happy* inputs $\mathcal{G}$ is a subproblem of both Strict TaRDiS and Nonstrict TaRDiS.

Happy TaRDiS
*Input:* A happy temporal graph $\mathcal{G} = (V, E, \lambda)$ and an integer $k$.
*Question:* Does $\mathcal{G}$ admit a TaRDiS of size at most $k$?

MaxMinTaRDiS is an extension of our problem in which we look to find a temporal assignment such that no TaRDiS of cardinality less than $k$ exists. As seen in Figure 3.3, scheduling social events is a natural combinatorial problem. In a similar manner to how Edge Colouring corresponds straightforwardly to scheduling meetings between one pair of people at a time to avoid a scheduling conflict, the MaxMinTaRDiS problem can be thought of as scheduling interactions (potentially simultaneously) so that the risk of widespread contagion is limited.

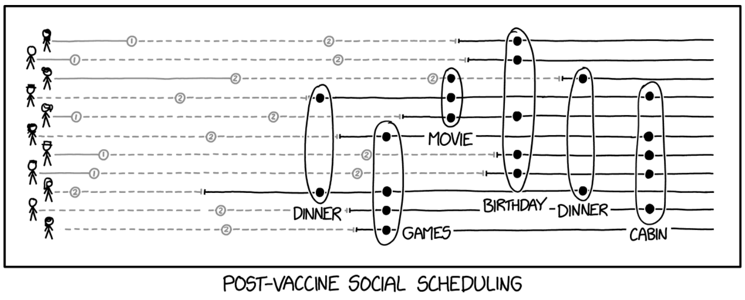


Figure 3.3: xkcd#2450 [86] depicts the scheduling of social events to minimize the risk of contagion.

(Strict/Nonstrict) MaxMinTaRDiS
*Input:* A static graph $H = (V, E)$ and integers $k$ and $\tau$.
*Question:* Does there exists a temporal assignment $\lambda : E \to 2^{[\tau]} \setminus \{\emptyset\}$ such that every strict/nonstrict TaRDiS admitted by $(H, \lambda)$ is of size at least $k$?

Likewise, the variant of this problem in which the temporal assignment $\lambda$ is required to be happy is referred to as Happy MaxMinTaRDiS. Note this is not a subproblem of Strict MaxMinTaRDiS or Nonstrict MaxMinTaRDiS. We will later show that Happy TaRDiS generalizes Edge Colouring (and hence conflict-free scheduling in our earlier analogy).

Happy MaxMinTaRDiS

*Input:* A static graph $H = (V, E)$ and integers $k$ and $\tau$.

*Question:* Does there exists a happy temporal assignment $\lambda : E \to [\tau]$ such that every TaRDiS admitted by $(H, \lambda)$ is of size at least $k$?

In this work, by "each variant" we refer to the Strict, Nonstrict and Happy variants of the problems.

### 3.1.2 Our Contribution

Our work highlights the complexity intrinsic to the dynamic behavior of spreading processes as soon as time-varying elements are incorporated into natural models. At a high level, we identify the minimum lifetime $\tau$ for which each problem is computationally hard. The existence of hardness results even with bounded lifetime justifies the need for parameters other than $\tau$ to obtain tractability results. We provide a fixed-parameter tractable (fpt)[1] algorithm for TaRDiS with parameters $\tau$ and the treewidth[2] of the footprint graph $\text{tw}(\mathcal{G}_\downarrow)$, and show existence of such an fpt algorithm for MaxMinTaRDiS with parameters $\tau$, $\text{tw}(\mathcal{G}_\downarrow)$, and $k$.

| $\tau$ | TaRDiS | | | MaxMinTaRDiS | | |
|---|---|---|---|---|---|---|
| | Strict | Nonstrict | Happy | Strict | Nonstrict | Happy |
| 1 | NP-c (Cor. 3.7) | Linear (Lem. 3.8) | Linear (Lem. 3.8) | coNP-c (Cor. 3.31) | Linear (Lem. 3.29) | Linear (Lem. 3.29) |
| 2 | | NP-c (Thm. 3.15) | | | NP-c (Cor. 3.50) | |
| 3 | | | NP-c (Thm. 3.12) | | $\in \Sigma_2^P$ (Lem. 3.25) | $\Sigma_2^P$-c (Thm. 3.42) |
| $\geq 4$ | | | | | | coNP-h $\cap\ \Sigma_2^P$ (Cor. 3.28, Lem. 3.25) |

Table 3.2: Computational complexity of our problems and its dependence on $\tau$.

Our results relating lifetime and computational complexity are highlighted in Table 3.2, and our parameterized complexity results are summarized in Table 3.3. For the case of happy temporal graphs, we exactly characterize the complexity of both TaRDiS and MaxMinTaRDiS with lifetime $\tau \leq 3$. Both problems are trivially solvable in linear time for $\tau \leq 2$. We show NP-completeness of Happy TaRDiS and $\Sigma_2^P$-completeness of Happy MaxMinTaRDiS when $\tau = 3$ - even when restricted to planar inputs of bounded degree.

For MaxMinTaRDiS, membership of NP is nontrivial since the existence of a polynomial-time verifiable certificate is uncertain. Indeed, the $\Sigma_2^P$-completeness of Happy MaxMinTaRDiS indicates no such certificate exists in general unless the Polynomial Hierarchy collapses. Interestingly, we show equivalence[3] of Nonstrict MaxMinTaRDiS restricted to inputs where $\tau = 2$ and Distance-3 Independent

[1] We use fpt (lowercase) as a descriptor for algorithms witnessing the inclusion of a problem in the parameterized complexity class FPT. A full definition is given in Section 3.4.

[2] Informally, the treewidth of a graph is a measure of its likeness to a tree. A formal definition is given in Section 3.4.

[3] Our definition of equivalence can be found in Section 3.3.

| | TaRDiS | | | MaxMinTaRDiS | | |
|---|---|---|---|---|---|---|
| Parameter | Strict | Nonstrict | Happy | Strict | Nonstrict | Happy |
| $\Delta + \tau$ | para-NP-h (Cor. 3.7) | para-NP-h (Thm. 3.15) | para-NP-h (Thm. 3.12) | para-NP-h (Cor. 3.31) | para-NP-h (Cor. 3.50) | para-NP-h (Thm. 3.42) |
| #LEE | n/a | FPT (Lem. 3.52) | | n/a | | |
| $k$ | W[2]-h (Cor. 3.7) | W[2]-h (Thm. 3.15) | W[2]-h (Cor. 3.17) | co-W[2]-c (Cor. 3.31) | W[1]-h (Cor. 3.50) | para-NP-h (Cor. 3.28) |
| $k + \tau$ | W[2]-h (Cor. 3.7) | W[2]-h (Thm. 3.15) | FPT (Lem. 3.53) | co-W[2]-c (Cor. 3.31) | W[1]-h (Cor. 3.50) | para-NP-h (Cor. 3.28) |
| $k + \tau + \Delta$ | FPT (Lem. 3.53) | W[2]-h (Thm. 3.15) | FPT (Lem. 3.53) | FPT (Lem. 3.62) | ??? | para-NP-h (Cor. 3.28) |
| $\text{tw}(\mathcal{G}_\downarrow) + \tau$ | FPT (Thm. 3.61) | | | ??? | | |
| $\text{tw}(\mathcal{G}_\downarrow) + \tau + k$ | FPT (Thm. 3.61) | | | FPT (Thm. 3.64) | | |

Table 3.3: Summary of our parameterized complexity results. Parameters are: maximum degree of the footprint graph $\Delta$, lifetime $\tau$, number of (weakly) locally earliest edges #LEE, input $k$, and treewidth of the footprint graph $\text{tw}(\mathcal{G}_\downarrow)$. Problems which are W[1]-, (co-)W[2]-, or para-NP-hard are ones for which there presumably exists no fpt algorithm.

Set, which is NP-complete [87], in Section 3.3.3. Given the uncertain membership of NP for the general problem, NP-completeness of this restriction of Nonstrict MaxMinTaRDiS indicates it is possibly *easier* than the unrestricted problem.

Having shown $\tau$ and planarity of the footprint graph alone are insufficient for tractability, we give an algorithm which solves TaRDiS on temporal graphs where the footprint is a tree in time polynomial in the number of time-edges $|\mathcal{E}|$ in the input graph. In addition, we give an fpt-algorithm for TaRDiS on nice tree decompositions [88]. This gives us tractability with respect to lifetime and treewidth of the footprint of the input graph combined. We also show existence of an algorithm for MaxMinTaRDiS which is fixed-parameter tractable with respect to $\tau$, $k$, and treewidth. This is achieved by applying Courcelle's theorem [89].

### 3.1.3 Related Work

Reachability and connectivity problems on temporal graphs have drawn significant interest in recent years. These have been studied in the context of network design [68, 17, 18] and transport logistics [19] (where maximizing connectivity and reachability at minimum cost is desired), and the study of epidemics [21, 22, 23, 24] and malware spread [25] (where it is not).

In research on networks, broadcasting refers to transmission to every device. In a typical model, communication rather than computation is at a premium, and there is a single source in a graph which does not vary with time [90]. Broadcasting-based questions deviating from this standard have been studied as well. Namely, there is extensive study of the complexity of computing optimal broadcasting schedules for one or several sources [91, 92], broadcasting in ad-hoc networks or time-varying graphs [93], and the choice of multiple sources (originators) for broadcasting in minimum time in a static graph [94, 95]. Ours is the first work to focus on the hardness of choosing multiple sources in a temporal graph to minimize the number of sources, in an offline setting.

One metric closely related to a temporal graph's vulnerability to contagion is its *maximum reachability*; that is, the largest number $k$ such that some vertex reaches $k$ vertices in the graph. In Enright, Meeks and Skerman [23] and Enright, Meeks, Mertzios and Zamaraev's [22] works, the problems of deleting and reordering edges in order to minimize $k$ are shown to be NP-complete. Problems on temporal graphs are often more computationally complex than their static counterparts [16, 96, 97, 98]. In such cases, efficient algorithms may still be obtained on restricted inputs. In work exploring disease spread through cattle, Enright and Meeks find that these real-world networks naturally have low treewidth [99]. Treewidth and other structural parameters have been used with varying degrees of success to give tractability on some of these temporal problems [16, 100, 101, 97].

A powerful tool in parameterized complexity is Courcelle's theorem, which gives tractability on graphs of bounded treewidth for problems that can be represented in monadic second order logic [102]. Unfortunately, there are many temporal problems which remain intractable even when the underlying graph is very strongly restricted, for example when it is a path, a star, or a tree [16, 96, 97, 98], all of which have treewidth 1. When this is the case, it is sometimes sufficient to additionally bound the lifetime of the temporal graph in addition to its underlying structure. This motivates our use of treewidth in combination with lifetime as parameters for the study of our problems.

The problems we study generalize two classical combinatorial graph problems, namely Dominating Set and Distance-3 Independent Set (D3IS). For a static graph $G = (V, E)$, a dominating set is a set of vertices $S$ such that every vertex is either adjacent to a vertex in $S$ or is in $S$ itself, and a D3IS is a set of vertices $S$ such that no pair of vertices $u, v \in S$ has a common neighbour $w$ (so all pairs are at distance at least three). The corresponding decision problems ask, for an integer $k$, if there exists a dominating set (respectively D3IS) of size at most (resp. at least) $k$, and are W[2]- (resp. W[1]-) complete [103, 87]; that is, even if $k$ is fixed it is unlikely there exists an algorithm solving the problem with running time $f(k) \cdot n^{O(1)}$. However, both Dominating Set and D3IS are fixed-parameter tractable parameterized by treewidth [104, 88][4]. Eto, Guo and Miyano [87] show that D3IS is NP-complete even on planar, bipartite graphs with maximum degree 3. They also show D3IS to be $W[1]$-hard on chordal graphs with respect to the size of the distance-3 independent set. D3IS is also shown to be APX-hard on $r$-regular graphs for all integers $r \geq 3$ and admit a PTAS on planar graphs by Eto, Ito, Liu, and Miyano [105]. Agnarsson, Damaschke and Halldórsson [106] show that D3IS is tractable on interval graphs, trapezoid graphs and circular arc graphs.

TaRDiS is exactly the problem of solving the directed variant of Dominating

[4]In [88], the more general Distance-$d$ Independent Set is called Scattered Set.

Set on a reachability graph [80]. The *reachability graph* of a temporal graph $\mathcal{G}$ is the static directed graph on the same vertex set as $\mathcal{G}$ with as edges those ordered pairs $(u, v)$ such that $u$ reaches $v$ in $\mathcal{G}$, and is shown to be efficiently computable by Whitbeck, Amorim, Conan, and Guillaume [77]. Temporal versions of dominating set and other classical covering problems have been well studied [68, 82, 107, 108], however these interpretations do not allow a chosen vertex to dominate beyond its neighbours. Furthermore, many other problems looking to optimally assign times to the edges of a static graph have been studied [66, 23, 74]. TaRDiS also generalises Temporal Source, which asks whether a single vertex can infect every other vertex in the graph, which is equivalent to the graph being a member of the class $\mathcal{J}^{1\forall}$ [82] mentioned earlier. There has also been extensive research into modifying an input temporal graph (subject to some constraints) to achieve some desired reachability objective [82, 84, 73, 85]. Deligkas, Eiben, and Skretas [84] have the objective of modifying $\lambda$ through a delaying operation so that *each vertex* in a designated set reaches all vertices in the graph; that is $\bigcap_{u \in S} R_u = V(\mathcal{G})$. Molter, Renken and Zschoche [73] consider both delay and deletion operations to modify $\lambda$ with the objective of minimizing the cardinality of the set $\bigcup_{u \in S} R_u$. Contrast these with our problem MaxMinTaRDiS, which has the objective of maximizing the minimum cardinality of **any** set $S$ satisfying $\bigcup_{u \in S} R_u = V(\mathcal{G})$.

### 3.1.4 Organization

This paper is organised as follows. We begin with classical complexity results for TaRDiS in Section 3.2. This consists of some preliminary observations pertaining to our problems in Section 3.2.1, which also convey some of the intuition for TaRDiS and provide tools for later technical results. Here we also consider temporal graphs with very small lifetime, showing that Nonstrict TaRDiS and Happy TaRDiS are efficiently solvable in temporal graphs with lifetime $\tau = 1$ and $\tau = 2$ respectively. In Sections 3.2.2 and 3.2.3 we show that these are the largest lifetime for which the problem is efficiently solvable in general. Finally, in Section 3.2.4, we show that TaRDiS is efficiently solvable when the footprint of the temporal graph is a tree.

Section 3.3 presents classical complexity results for our MaxMinTaRDiS problems. We again begin with some preliminary results in Section 3.3.1 which provide the scaffolding for our later proofs as well as some easy bounds against which we compare our more technical results. This section allows us to draw the comparison between our problems and the classical problems Edge Colouring and Dominating Set. The remainder of this section includes our proofs of the contrasting results that Nonstrict MaxMinTaRDiS generalizes D3IS (and is in NP when restricted to $\tau = 2$), and of the $\Sigma_2^P$-completeness of Happy MaxMinTaRDiS.

Having established the hardness of both problems in the general case, and provided an algorithm for TaRDiS when the footprint graph is a tree, we turn to parameterized complexity in Sections 3.4 and 3.5. The majority of these sections focus on the structural parameter of treewidth of the footprint graph. We give an algorithm that solves TaRDiS on a nice tree decomposition of a graph in Section 3.4.3. In Section 3.5.1 we show existence of such an algorithm for MaxMinTaRDiS by leveraging Courcelle's theorem. Finally, we give some concluding remarks and future research directions in Section 3.6.

## 3.2 Classical complexity results for TaRDiS

In this section, we establish NP-completeness of each variant of TaRDiS, and characterize the maximum lifetime $\tau$ for which the problem is tractable. We also give an algorithm solving the problem in polynomial time when the footprint graph $\mathcal{G}_\downarrow$ is a tree. A more general algorithm for footprint graphs of bounded treewidth is later given in Section 3.4.

### 3.2.1 Containment in NP, useful tools, and small lifetime

We begin this section by showing containment of Strict, Nonstrict and Happy TaRDiS in NP and introducing the notions of a (weakly) locally earliest edge and (weakly) canonical TaRDiS. We then show that there always exists a canonical minimum TaRDiS in happy temporal graphs. This allows us to reduce the number of cases we must consider when solving Happy MaxMinTaRDiS in Section 3.3.2. Finally, we consider the restrictions of the lifetime to $\tau = 1$ and $\tau \leq 2$, where Nonstrict TaRDiS and Happy TaRDiS respectively are easily shown to be efficiently solvable. We later show that these are the largest values of $\tau$ for which these problems are tractable.

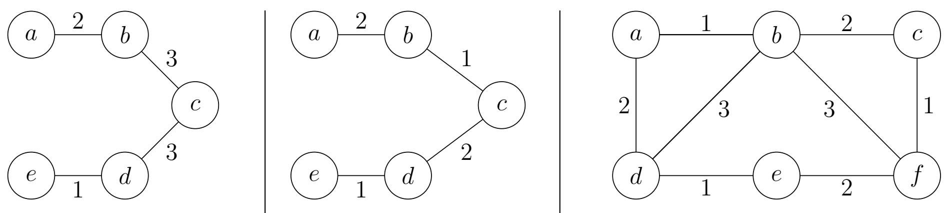
**Lemma 3.2.** Each variant of TaRDiS is in NP.

*Proof.* The reachability set of a vertex can be computed in polynomial time (e.g. by a modification of breadth-first search – see [109, 66, 110]). Therefore, we can verify whether a set temporal reachability dominates a temporal graph in polynomial time. □

We now introduce the notion of a canonical TaRDiS. The following results will then allow us to make assumptions about the composition of a minimum TaRDiS when working with proper or happy temporal graphs, or on Nonstrict TaRDiS.

**Definition 3.3** ((Weakly) locally earliest, (weakly) canonical TaRDiS)**.** In a temporal graph $\mathcal{G}$, a time-edge $((u, v), t)$ is *locally earliest* if every other time-edge incident to either $u$ or $v$ is at a time $t' > t$. If the weaker constraint $t' \geq t$ holds, then we call the time-edge weakly locally earliest. We say an edge $(u, v)$ is (weakly) locally earliest if, for some $t$, the time-edge $((u, v), t)$ is (weakly) locally earliest. A (weakly) *canonical TaRDiS* consists exclusively of vertices which are incident to a (weakly) locally earliest edge.

In Figure 3.4b, each of $\{b, d\}$, $\{b, c, d\}$ and $\{a, c, e\}$ is a TaRDiS, but only the former two are canonical.



(a) $R_a^{\leq} \neq R_a^{<}$ and $R_e^{\leq} \neq R_e^{<}$.

(b) $\{a, e\}$ is not canonical.

(c) Every TaRDiS is canonical.

Figure 3.4: Three temporal graphs, all admitting $\{a, e\}$ as a minimum TaRDiS.

**Lemma 3.4.** In a temporal graph, there always exists a minimum weakly canonical nonstrict TaRDiS. In a proper temporal graph, there always exists a minimum canonical TaRDiS.

*Proof.* We may assume without loss of generality that $\mathcal{G}_\downarrow$ is a connected graph. Note that a weakly locally earliest edge in a proper temporal graph is necessarily a locally earliest edge. Consequently, for the remainder of the proof, we focus on weakly locally earliest edges.

Given a TaRDiS $S$, we can find a weakly canonical TaRDiS $S'$ by replacing vertices not incident to weakly locally earliest edges by vertices that are. To see this, suppose $x \in S$ is not incident to a weakly locally earliest edge. Then there exist vertices $u, v \in V(\mathcal{G})$ and time-edges $((x,u),t), ((u,v),t')$ such that $t > t'$. Suppose that $t$ is the earliest time that this is true for. Then, since all vertices reachable from $x$ must be temporally reachable from $v$ by a path appending the time-edges $((v,u),t'), ((u,x),t)$, $R_x \subseteq R_v$. Therefore, we can swap $x$ for $v$ without losing domination of the whole graph. We perform this replacement repeatedly for every such $x$ until we obtain a set of vertices each incident to at least one weakly locally earliest edge. This is precisely a weakly canonical TaRDiS. □

This gives us a result adjacent to a known result in temporal graph theory: define a set $C$ to be an $\mathcal{S}$-component if $C = R_u$ for some $u$ and there is no $v$ such that $C \subsetneq R_v$. Then Lemma 4 from [79] states that, in a temporal graph without isolated vertices, the number of $\mathcal{S}$-components is at most the number of locally earliest edges.

**Corollary 3.5.** The number of weakly locally earliest edges upper-bounds the size of a minimum nonstrict TaRDiS.

We now show that the classical problem Dominating Set is a special case of Strict TaRDiS. Dominating Set is known to be NP-complete, even on planar graphs with maximum degree $\Delta = 3$ (hence para-NP-hard with respect to $\Delta$) [111], and W[2]-hard with respect to the size of the dominating set [103].

**Lemma 3.6.** For all positive integers $\tau$ and instances $(G, k)$ of Dominating Set, a temporal graph $\mathcal{G}$ of lifetime $\tau$ can be found in linear time, where $(\mathcal{G}, k)$ is a yes-instance of Strict TaRDiS if and only if $(G, k)$ is a yes-instance of Dominating Set.

*Proof.* The construction of $\mathcal{G}$ can be seen in Figure 3.5. We append a path $P$ of length $\tau - 1$ to an arbitrary vertex $v$ in $G$. The temporal assignment of $\mathcal{G}$ is defined as follows: all edges in $E(G)$ are assigned time 1, and the edges in $P$ are assigned times in ascending order from $v$. It is clear that there exists a dominating set of cardinality $k$ in $G$ if and only if there exists a strict TaRDiS of cardinality $k$ in $\mathcal{G}$. □

We note that, in our construction, $k$ is unchanged and the maximum degree of $\mathcal{G}_\downarrow$ is bounded by $\Delta(G) + 1$; giving us the following corollary.

**Corollary 3.7.** Strict TaRDiS is NP-complete, W[2]-hard with respect to $k$, and para-NP-hard with respect to $\Delta + \tau$, where $\Delta$ is the maximum degree of the footprint graph and $\tau$ is the lifetime of the temporal graph.

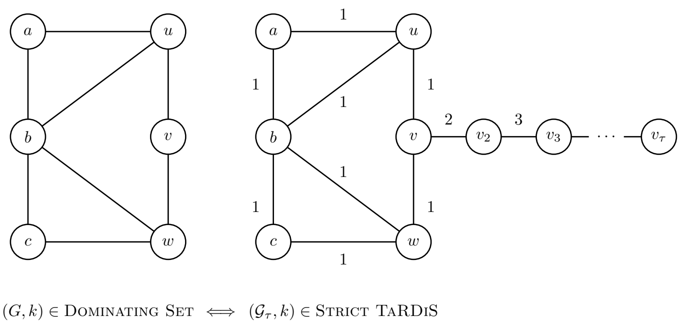

a
u
b
v
c
w
1
2
3
$v_2$
$v_3$
$\cdots$
$v_\tau$
$(G, k) \in$ Dominating Set $\iff$ $(\mathcal{G}_\tau, k) \in$ Strict TaRDiS

Figure 3.5: Illustration of the construction used to reduce from Dominating Set to Strict TaRDiS with arbitrary lifetime $\tau$.

We now consider the problems Nonstrict TaRDiS and Happy TaRDiS with very restricted lifetimes.

**Lemma 3.8.** Nonstrict TaRDiS can be computed in linear time when $\tau = 1$. When $\tau \leq 2$, Happy TaRDiS is solvable in linear time.

*Proof.* If every edge is active at the same time, every vertex can be temporally reached by a nonstrict path from any vertex in the same connected component. Therefore, if $\tau = 1$, we find a minimum TaRDiS by choosing exactly one vertex in every connected component of the graph.

If a temporal graph is happy and the lifetime of the graph is at most 2, the footprint of the graph must be 2-edge colourable, and so consist of only paths or even cycles. Any TaRDiS must include every vertex which has no neighbours. Then, for any path, we iteratively we choose a leaf, add its neighbour to the TaRDiS, and delete all vertices reachable from the TaRDiS – until the entire path has been deleted. We use a similar method on even cycles by simply picking an arbitrary vertex $v$ (by symmetry equivalent to any other on the same cycle) to be in the TaRDiS. From here, we find the subgraph induced by removing the vertices reachable from $v$, giving us a path to which we can apply the previous procedure. □

### 3.2.2 NP-completeness of Happy TaRDiS with lifetime 3

Above, we establish that Happy TaRDiS can trivially be solved in linear time when the input has lifetime $\tau \leq 2$ by Lemma 3.8. Here we show that the problem immediately becomes NP-complete for inputs where $\tau = 3$, even when $\mathcal{G}_\downarrow$ is planar.

We give a reduction from the problem Planar exactly 3–bounded 3–SAT, which asks whether the input Boolean formula $\phi$ admits a satisfying assignment. We are guaranteed that $\phi$ is a planar formula in 3-CNF (a disjunction of clauses each containing at most 3 literals) with each variable appearing exactly thrice and each literal at most twice. This problem is shown to be NP-hard by Tippenhauer and Muzler [112].

#### Construction

Let $\phi$ be an instance Planar exactly 3–bounded 3–SAT consisting of clauses $\{c_1, \dots, c_m\}$ over variables $X = \{x_1, \dots, x_n\}$. Let $l_{2i}$ (resp. $l_{2i-1}$) denote the positive (resp. negative) literal for variable $x_i$. We also introduce a special literal $\perp$ which is always `False`. We rewrite every 2-clause of $\phi$ to include the special literal $\perp$ (e.g. the clause $(x_i \vee \neg x_j)$ becomes $(x_i \vee \neg x_j \vee \perp)$ then $(l_{2i} \vee l_{2j-1} \vee \perp)$). Let $\phi'$ denote the new formula obtained by doing this. Note that $\phi'$ admits a satisfying assignment (in which $\perp$ evaluates to `False`) if and only if $\phi$ admits a satisfying assignment.

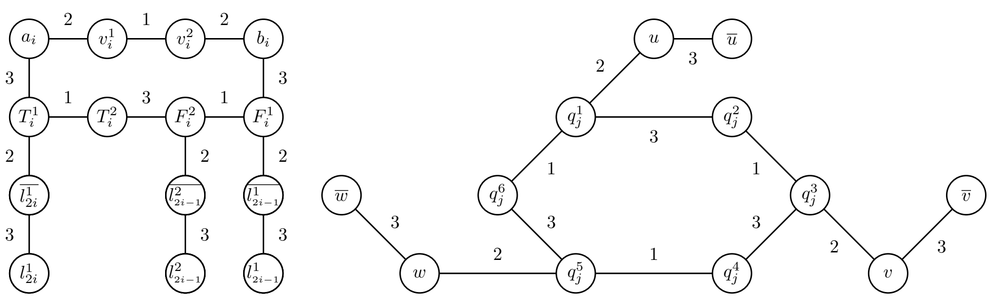


Figure 3.6: Gadgets and adjacent literal vertices in the reduction from Planar exactly 3–bounded 3–SAT to Happy TaRDiS. Left: the variable gadget for $x_i$, which appears twice negatively. Right: the clause gadget for the clause $c_j$.

In this construction $E(\mathcal{G}) = E_1 \cup E_2 \cup E_3$; $\lambda$ is implicitly defined on this basis. We introduce *literal vertices* as follows. Iterate over $\phi'$: at the $a$th appearance of each literal $l_i$, create two new vertices $l_i^a$ and $\overline{l_i^a}$, and let $(l_i^a, \overline{l_i^a}) \in E_3$. We refer to the set of all literal vertices as $L$. We say the vertex $l_i^a$ *belongs* to the clause $c_j$ in which $l_i$ appears for the $a$th time.

For each variable $x_i$, we introduce vertices $V_i = \{T_i^1, T_i^2, F_i^1, F_i^2, v_i^1, v_i^2, a_i, b_i\}$, connected in a cycle labeled as shown in Figure 3.6. Specifically, we have $(T_i^1, T_i^2), (F_i^1, F_i^2), (v_i^1, v_i^2) \in E_1$, $(a_i, v_i^1), (b_i, v_i^2) \in E_2$, and $(a_i, T_i^1), (b_i, F_i^1), (T_i^2, F_i^2) \in E_3$. Then, for each positive (resp. negative) literal vertex $\overline{l_{2i}^a}$ (resp. $\overline{l_{2i-1}^a}$), we add $(\overline{l_{2i}^a}, T_i^a)$ to $E_2$.

Every three literals vertices $u, v, w$ belonging to the same clause $c_j$ are connected with the *clause gadget* as shown in Figure 3.6. Namely, we introduce the vertices $Q_j = \{q_j^1, \dots, q_j^6\}$, with $(q_j^1, q_j^2), (q_j^3, q_j^4), (q_j^5, q_j^6) \in E_3$, $(q_j^2, q_j^3), (q_j^4, q_j^5), (q_j^6, q_j^1) \in E_1$, and $(q_j^1, u), (q_j^3, v), (q_j^5, w) \in E_2$. If, for example, clause $c_j = (l_5 \vee l_{17} \vee l_{20})$ is the first appearance of $l_5$ and $l_{20}$ and the second appearance of $l_{17}$ in $\Phi$, then $u = l_5^1$, $v = l_{17}^2$ and $w = l_{20}^1$. If $\perp \in \{u, v, w\}$ then we create a special dummy vertex for this clause and attach it as normal (a new dummy is created for each clause originally on two literals).

Lastly, we let $k = 2m + 2n$. This concludes our construction of the Happy TaRDiS instance $(\mathcal{G}, k)$.

### Properties of the construction

We will show that $\mathcal{G}$ admits a TaRDiS $S$ of size $k$ if $\phi$ is satisfiable, and that any TaRDiS is of size at least $k+1$ otherwise. It is worth noting that our construction preserves planarity of $\phi$. That is, $\mathcal{G}_\downarrow$ is a planar graph.

We now show that the existence of a TaRDiS of size at most $k$ implies that of a satisfying assignment.

**Lemma 3.9.** Any *canonical* TaRDiS of $\mathcal{G}$ includes at least 2 vertices from each variable gadget and at least 2 vertices from each clause gadget, and so has size at least $k$.

*Proof.* Recall that a *canonical* TaRDiS $S$ by definition is incident only to locally earliest edges. In the case of $\mathcal{G}$, none of the *literal vertices* can belong to $S$, i.e. $S \subseteq V \setminus L$ where $L$ is the set of literal vertices. For each variable $x_i$ and clause $c_j$, no vertex from $V_i$ reaches any vertex from $Q_j$, and vice versa. Note that, for all $j \in [m]$, every vertex in $Q_j$ reaches exactly 4 vertices in $Q_j$, for example $R_{q_j^3} = \{q_j^1, q_j^2, q_j^3, q_j^4\}$. Thus, $|S \cap Q_j| \geq 2$, else $S$ would not reach every vertex in $Q_j$. Similarly, for all $i \in [n]$, every vertex in $V_i$ reaches at most 6 variable gadget vertices, so $|S \cap V_i| \geq 2$. □

**Lemma 3.10.** If $\mathcal{G}$ admits a TaRDiS $S$ of size $k$ then $\phi$ is satisfiable.

*Proof.* Suppose $\mathcal{G}$ admits a TaRDiS $S$ of size at most $k$. We first apply Lemma 3.4, which allows us to assume that $S$ is canonical. Then, by Lemma 3.9, we have that $S$ is of size *exactly* $k$, and that for all $i$ and $j$, $|S \cap V_i| = 2$ and $|S \cap Q_j| = 2$.

We show that a satisfying assignment $X$ to the variables of $\phi$ can be obtained from $S$. For each variable $x_i$, assign $x_i =$ `True` if $T_i^1$ or $T_i^2 \in S$, and $x_i =$ `False` otherwise. Note that, since $S$ is a canonical TaRDiS, $a_i$ and $b_i$ are not in $S$. Hence $F_i^1 \in S$ or $F_i^2 \in S$ if and only if $x_i =$ `False` in our assignment. Suppose for contradiction there is some clause $c_j$ which is not satisfied. Then, under this assumption, the gadget $c_j$ is incident to three *literal vertices* $\{u, v, w\}$, none of which are reached from their respective *variable gadgets*. Since $S$ is a TaRDiS, $q_j^1$ or $q_j^6$ is in $S$ (as $u$ must be reached from within $Q_j = \{q_j^1, \ldots, q_j^6\}$), and likewise $q_j^2$ or $q_j^3$ and $q_j^4$ or $q_j^5$ are in $S$ (as $v$ and $w$ respectively must be reached from within $Q_j$). Hence $|Q_j \cap S| \geq 3$. By applying Lemma 3.9 we get that $|S| \geq 2m + 2n + 1 = k + 1$, a contradiction. Hence $X$ must satisfy $\phi$. □

**Lemma 3.11.** If $\phi$ is satisfiable then $\mathcal{G}$ admits a TaRDiS $S$ of size $k$.

*Proof.* Given a satisfying assignment to $\phi$, we construct $S$ as follows. We start with $S = \cup_i\{v_i^2\}$. Then if $x_i =$ `True` (respectively $x_i =$ `False`) under the assignment, add $T_i^1$ to $S$ (resp. $F_i^1 \in S$). Now $S$ has size $2n$ and a vertex in $S$ reaches the literal vertices for every literal set to `True` in the assignment, but none of the clause gadget vertices.

For each clause $c_j$, there is some literal in $c_j$ which is assigned `True`, and hence at least one literal vertex incident to $Q_j$ is reached from the corresponding variable gadget. Denote this literal vertex $u$, and the other literal vertices incident to $Q_j$ (which may or may not be reached from their respective variable gadgets) $v$ and $w$, respectively. We add the neighbours of $v, w$ in $Q_j$ to $S$. That is, $S = S \cup ((N[v] \cup N[w]) \cap Q_j)$. Now $S$ has size $2n + 2m$ and reaches: all variable gadget vertices; all clause gadget vertices; and all literal vertices (since every literal vertex is incident to a clause gadget). Hence $S$ is a TaRDiS of size exactly $k$. □

**Theorem 3.12.** Happy TaRDiS is NP-complete, even restricted to instances where the footprint of the temporal graph is planar and its lifetime is 3.

*Proof.* We have membership of NP from Lemma 3.2. The construction above produces an instance $(\mathcal{G}, k)$ of Happy TaRDiS from an instance $\phi$ of Planar Exactly 3–Bounded 3–SAT in polynomial time. Combining Lemmas 3.10 and 3.11 shows that $\mathcal{G}$ admits a TaRDiS of size at most $k$ if and only if $\phi$ is satisfiable. □

This result generalises to both Strict and Nonstrict TaRDiS, giving us NP-completeness of all variants with bounded lifetime and a planar footprint.

### 3.2.3 NP-completeness of Nonstrict TaRDiS with lifetime 2

We show hardness of Nonstrict TaRDiS by reducing from Set Cover, which is known to be NP-complete [10] and W[2]-hard with respect to the parameter $k$ [88]. The Set Cover problem is defined as follows.

Set Cover
*Input:* Universe $\mathcal{U} = \{x_1, \dots x_n\}$, a family $\mathcal{S} = \{s_i | s_i \subseteq \mathcal{U}\}$ of subsets in $\mathcal{U}$ and an integer $k$.
*Question:* Is there a set $\{s_{i\in I}\}$ with at most $k$ elements such that $\cup_{i\in I} s_i = \mathcal{U}$?

Given an instance of Set Cover, we build a temporal graph $\mathcal{G}$ with vertex set $V(\mathcal{G}) = \mathcal{U} \cup \mathcal{S} \cup \{a_i^j | \; \exists x_i \in \mathcal{U}, s_j \in \mathcal{S} : x_i \in s_j\}$. To this vertex set we add the edges:

- connecting $s_j$ in a path at time 1 to all $a_i^j$ for all $i$ such that $a_i^j$ exists;
- connecting $x_i$ in a path at time 2 to all $a_i^j$ for all $j$ such that $a_i^j$ exists;
- $(s_j, s_{j+1})$ at time 2 for all $j \in [1, m-1]$.

A sketch of $\mathcal{G}$ can be found in Figure 3.7. Note that the maximum degree of $\mathcal{G}$ is 4. We keep the same value of $k$ in our construction; that is, we construct an instance $(\mathcal{G}, k)$ of Nonstrict TaRDiS from the instance $(\mathcal{U}, \mathcal{S}, k)$ of Set Cover.

**Lemma 3.13.** Any TaRDiS in $\mathcal{G}$ can be reduced to a TaRDiS of the same or smaller size in $V(\mathcal{G}) \cap \mathcal{S}$.

*Proof.* Suppose there is a TaRDiS $S$ in $\mathcal{G}$ containing a vertex $v \in V(\mathcal{G}) \setminus \mathcal{S}$. We have two cases:

1. $v = x_i$ for some $1 \leq i \leq n$, or
2. $v = a_i^j$ for some $1 \leq i \leq n$ and $1 \leq j \leq m$.

In the first case, we can swap $x_i$ for any neighbour (or delete $x_i$ if $N[x_i] \subset S$). This must be a vertex $a_i^j$ for some $1 \leq j \leq m$. Since the reachability set of $a_i^j$ is a strict superset of that of $x_i$, $S$ remains a TaRDiS when $x_i$ is replaced by its neighbour. This leaves us with case 2. The reachability sets of $a_i^j$ are equal to the reachability set of $s_j$ for all $a_i^j \in V(\mathcal{G}) \setminus (\mathcal{U} \cup \mathcal{S})$. Therefore, we can replace vertices in $S$ with those corresponding to sets in $\mathcal{S}$ without changing the reachability set of the TaRDiS. □

**Lemma 3.14.** The temporal graph $\mathcal{G}$ admits a TaRDiS of cardinality $k$ if and only if $(\mathcal{U}, \mathcal{S}, k)$ is a yes-instance of Set Cover.

*Proof.* Suppose there is a set $\{s_{j\in I}\}$ of cardinality $k$ which is a set cover of $\mathcal{U}$. Then we claim the same set $S$ is a TaRDiS in $\mathcal{G}$. If an element $x_i$ is in the set $s_j$, then by construction of $\mathcal{G}$, there is a path from $s_j$ to $x_i$ via $a_i^j$. Furthermore, all vertices $s_{j'}$ for $1 \leq j' \leq m$ are temporally reachable from any $s_j$. Since all $x_i$ appear in at least one of $\{s_{j\in I}\}$, we know that all $x_i$ and all $s_j$ are temporal reachability dominated by $S = \{s_{j\in I}\}$. What remains to check is that all $a_i^j$ are also reached by a vertex in $S$. Each $a_i^j$ is connected to $x_i$ at time 2. Therefore, since all $x_i$ are reached at time 2 from $S$, each $a_i^j$ must also be temporally reachable from $S$.

Now suppose that there is a TaRDiS $S$ of cardinality $k$ of $\mathcal{G}$. By Lemma 3.13, we can assume that $S \subseteq \mathcal{S}$. Observe that there is a nonstrict temporal path from a vertex $s_j$ that reaches a vertex $x_i$ if and only if $x_i \in s_j$. Hence, a vertex $x_i$ is temporally reachable from some vertex in $S$ if and only if a set $s_j$ containing $x_i$ is in $S$. Since every $x_i$ is reachable from some vertex in $S$, then, $S$ is a set cover. □

We note that since we use the same input $k$ for both problems, Nonstrict TaRDiS is $W[2]$-hard with respect to $k$. This proves the following theorem.

**Theorem 3.15.** For every lifetime $\tau \geq 2$ Nonstrict TaRDiS is NP-complete and W[2]-hard with respect to $k$. This holds even on graphs with maximum degree 4.

We now extend our construction to obtain W[2]-hardness of Happy TaRDiS. Consider the following construction for a happy temporal graph $\mathcal{G}'$. Begin with the

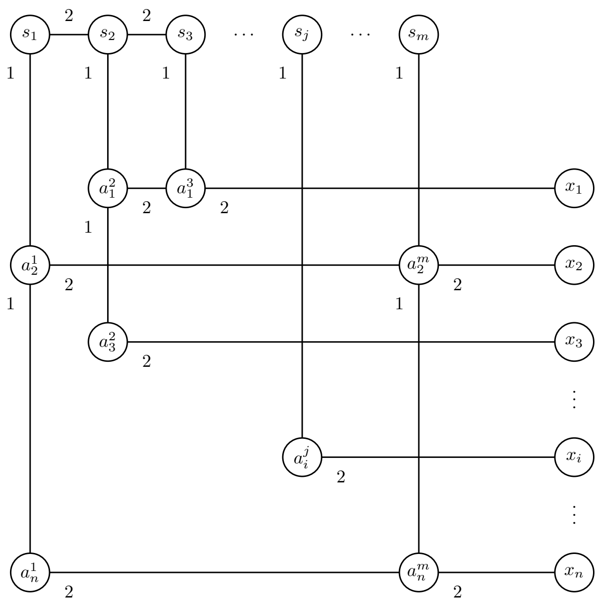


Figure 3.7: Sketch for the reduction from Set Cover to Nonstrict TaRDiS

temporal graph $\mathcal{G}$ as constructed earlier in Figure 3.7. Then, for each $i$, add edges such that the set $\{s_i\} \cup \bigcup_j a_j^i$ forms a clique. For each $j$, we also add edges such that the set $\{x_j\} \cup \bigcup_i a_j^i$ forms a clique. Likewise, add edges so that $\cup_j s_j$ forms a clique. Then we order all time-edges arbitrarily such that all edges in each clique containing edges formerly in $E_1$ are active before the edges in each clique containing all time-edges formerly in $E_2$. This concludes the description of $\mathcal{G}'$. Observe that $\mathcal{G}'$ is a happy temporal graph, since each snapshot contains just one edge.

**Lemma 3.16.** The temporal graph $\mathcal{G}'$ admits a TaRDiS of cardinality $k$ if and only if $(\mathcal{G}, k)$ is a yes-instance of Nonstrict TaRDiS.

*Proof.* We show that a vertex $u$ temporally reaches a vertex $v$ in $\mathcal{G}$ if and only if $u$ also reaches $v$ in $\mathcal{G}'$. Thus, proving that $\mathcal{G}'$ admits a TaRDiS of cardinality $k$ if and only if $\mathcal{G}$ admits a TaRDiS of cardinality $k$.

Suppose we have a path $p$ from a vertex $u$ to a vertex $v$ in $\mathcal{G}$. We have two cases; namely, the edges in the path are assigned the same time in $\mathcal{G}$ or they are not. In the former, there must still be a (one-edge) path from $u$ to $v$ in $\mathcal{G}'$ since we have added edges such that all connected components in either snapshot of $\mathcal{G}$ form a clique. In the latter, we can break $p$ into a path at time one $p_1$ and a path $p_2$ at time two. These paths can each be replaced by an edge as in the case where $p$ consists of edges which occur at the same time. Therefore, for every path in $\mathcal{G}$, there exists a path with the same starting and terminal vertex in $\mathcal{G}'$.

Now suppose we have a path $p'$ in $\mathcal{G}'$ from $u'$ to $v'$. Since we have cliques for each connected component in the snapshots of $\mathcal{G}$, the path consisting of fewest edges in $\mathcal{G}'$ from $u'$ to $v'$ must consist of at most 2 edges. Suppose without loss of generality that $p'$ is the shortest path from $u'$ to $v'$. If $p'$ consists of a single edge, then $u'$ and $v'$ must be in the same connected component of a snapshot of $\mathcal{G}$, thus there exists a path from $u'$ to $v'$ in $\mathcal{G}$. Now suppose that $p'$ consists of two edges. We know that $v'$ and $u'$ are not adjacent in $\mathcal{G}'$, else $p'$ would not be the shortest path from $u'$ to

$v'$. Therefore, $u'$ and $v'$ must be members of two different cliques in $\mathcal{G}'$. Let $w'$ be the vertex adjacent to both $u'$ and $v'$ which $p'$ traverses. Then, since $u'$ and $w'$ share an edge which is earlier than the edge $(w', v')$, they must be in the same connected component of $\mathcal{G}_1$. Similarly, $w'$ and $v'$ must be in the same connected component of $\mathcal{G}_2$. Therefore, there is a path from $u'$ to $v'$ in $\mathcal{G}$. Hence, a vertex $u$ temporally reaches a vertex $v$ in $\mathcal{G}$ if and only if $u$ also reaches $v$ in $\mathcal{G}'$. □

**Corollary 3.17.** Happy TaRDiS is NP-complete and W[2]-hard with respect to $k$.

### 3.2.4 Algorithm for TaRDiS on Trees

In this section, we show that each variant of TaRDiS can be solved in polynomial time on inputs which are trees, even when $k$ and lifetime are unbounded. We introduce some preliminary notions needed for the algorithm.

In this section, we deal with *non-simple* temporal graphs; each edge $e \in E(\mathcal{G})$ may appear several times. We define the functions $\lambda_{\min} : E \to [\tau]$ and $\lambda_{\max} : E \to [\tau]$ which return the earliest and latest appearance of an edge, respectively.

In the following we refer to *rooted trees.* A rooted tree is a tree $T = (V, E)$ with a special vertex $r$; the *root.* A vertex $w$ is a *descendant* of a vertex $v$ if $v$ lies on the unique path from $w$ to $r$. We refer to the vertices neighbouring $v$ which are descendants of $v$ as its *children.* If a vertex $w$ is a child of $v$, then $v$ is its *parent.* Similarly, the parent of a parent vertex is referred to as a *grandparent.* The subgraph induced by a vertex $v$ and its descendants is referred to as the *subtree rooted at* $v$. The input of the algorithm is a *rooted temporal tree* $\mathcal{G}$. By this, we mean a temporal graph whose footprint $\mathcal{G}_\downarrow$ is a rooted tree. We define the function $C(p)$ to return the set of children for a parent vertex $p$. We refer to the distance of a vertex in the tree from the root as its *depth.*

#### Intuition

We now give a high-level description of how Algorithm 1 works to aid the reader in following the pseudocode we provide. We initialise a counter $k$ and two empty sets. The first set $S$ will become a TaRDiS if a TaRDiS of size $k$ exists. The second set $M$ is a set of vertices which are temporally reachable from those in $S$ or will be temporally reachable from a later vertex added to $S$. Roughly, these are the vertices we do not need to worry about. We refer to them as *marked.* Vertices not in $M$ will be referred to as *unmarked.*

We aim to select the highest possible ancestor from each leaf which covers it. Once a vertex is selected to be in the TaRDiS, we mark everything temporally reachable from that vertex and look for a new leaf to work from in the subgraph induced by the unmarked vertices. We also use marking a vertex to ignore any leaves which will be reached from an ancestor if their parent is.

We work from the leaves of $\mathcal{G}_{\downarrow}$ to the root $r$. At each iteration, we either mark vertices or reduce the number of appearances of an edge. We continue until the unmarked graph is a star or empty. The star graph is the complete bipartite graph $K_{1,l}$. Its central vertex is a vertex incident to every edge of the graph if $l \geq 1$, and is the sole vertex in the graph otherwise.

If a vertex $v$ with parent $p$ and grandparent $g$ is reachable from its grandparent using the latest times on the edges $(v, p)$ and $(p, g)$ respectively, it will be reached by a vertex in $S$ if its parent is. Therefore, we can mark that vertex. If a vertex $v$ is an unmarked vertex of maximum depth and not temporally reachable from its grandparent, we must add a vertex in $N[v]$ to $S$. Finally, if $v$ is reachable from $g$ but the last appearance of $(v, p)$ is strictly before the last appearance of $(p, g)$, then we will never use that appearance of $(p, g)$ to reach $v$. Since we have selected $v$ to be the child of $p$ such that the time of the latest appearance of $(v, p)$ is minimised, we can remove the last appearance of $(p, g)$ without changing the reachability of $v$ and its siblings from $g$. Figure 3.8 illustrates an execution of our algorithm, in which the first (respectively second, third, fourth, fifth) iteration satisfies the conditions on line 12 (resp. lines 16, 19, 16 –again–, and 5) of our algorithm. Because the input is a proper temporal graph the execution is identical for the strict and non-strict settings. We solve Strict TaRDiS or Nonstrict TaRDiS using the variable $s$ which depends on the Boolean flag `Strict`.

**Algorithm 1** TaRDiS on Trees

**Input:** A rooted temporal tree $\mathcal{G}$ and a Boolean flag `Strict`.
**Output:** A TaRDiS $S$ of minimum size.

```
1:  Initialise $S = \emptyset, M = \emptyset$, and $s = 0$.
2:  if Strict then
3:      Set $s = 1$.
4:  while $M \neq V(\mathcal{G})$ do
5:      if $\mathcal{G} \setminus M$ is a star then
6:          Add the central vertex to $S$ and return $S$.
7:      else Denote by $L$ the set of vertices in $V(\mathcal{G}) \setminus M$ at maximum depth.
8:          Let $p$ be the parent of an arbitrary vertex from $L$.
9:          Choose the vertex $l \in C(p) \setminus M$ which minimizes $\lambda_{\max}(p, l)$.
10:         Let $g$ be the parent of $p$ (and hence the grandparent of $l$).
11:         while $l$ is not in $M$: do
12:             if $\lambda_{\max}(l, p) < \lambda_{\min}(p, g) + s$ then
13:                 Add $p$ to $S$.
14:                 Find $R_p$.
15:                 Add $R_p$ to $M$: $M = M \cup R_p$.
16:             else if $\lambda_{\max}(l, p) \geq \lambda_{\max}(p, g) + s$ then
17:                 $M := M \setminus \{p\}$ (note this does nothing if $p \notin M$ already).
18:                 Add all children of $p$ to $M$.
19:             else do $\lambda(p, g) = \lambda(p, g) \setminus \{\lambda_{\max}(p, g)\}$.
20: Return $S$.
```

**Lemma 3.18.** The algorithm `TaRDiS on Trees` always terminates in time $O(|\mathcal{E}|^2)$, where $\mathcal{E}$ is the set of time-edges in $\mathcal{G}$.

*Proof.* The algorithm operates by adding vertices to a set $S$ until the whole tree is marked. Observe that each iteration of the inner while-loop (line 11) results in either

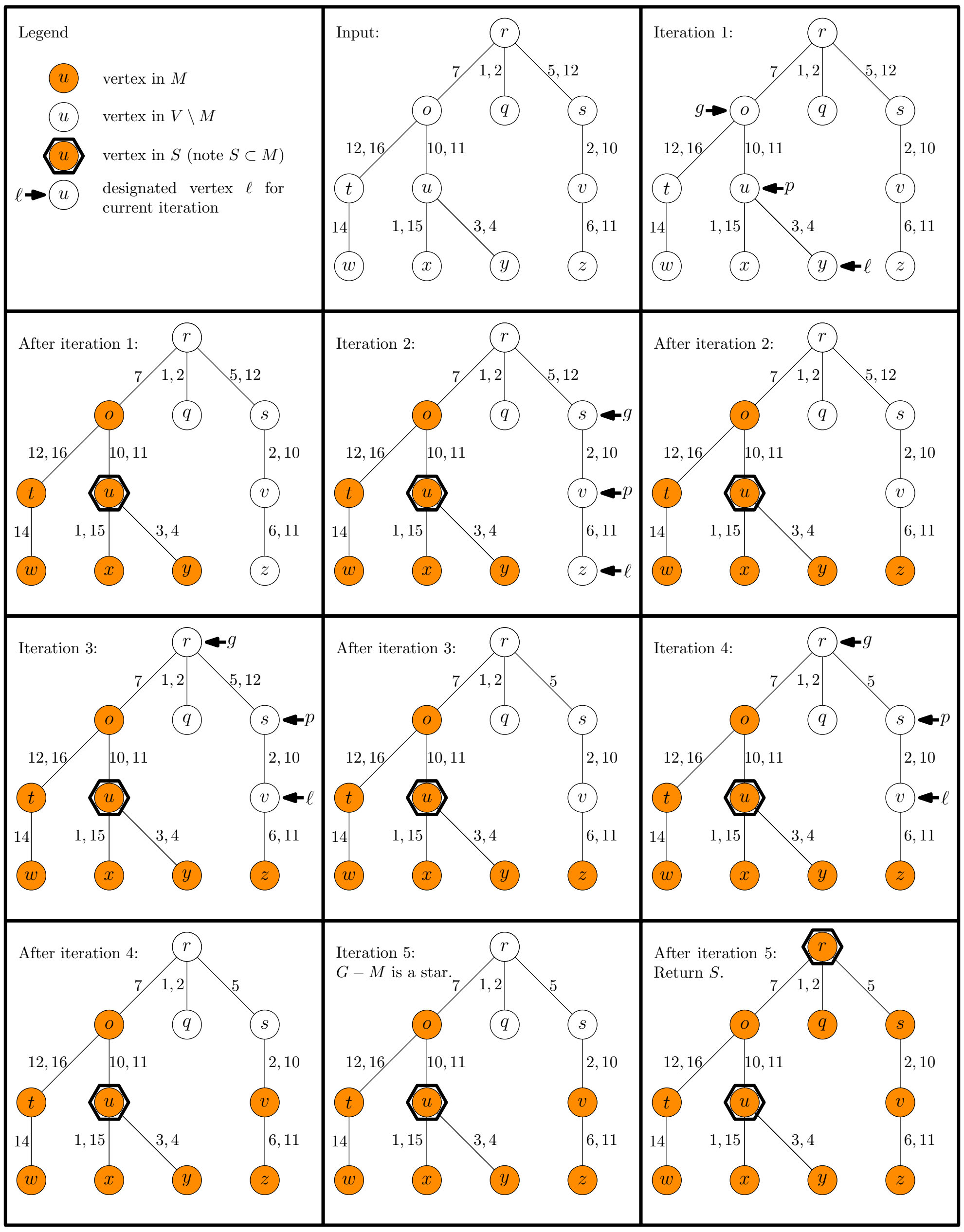


Figure 3.8: An example of an execution of Algorithm 1.

an increase in size of the set $M$ of marked vertices or removal of an appearance of the time-edge between parent and grandparent vertices of the vertex in question. Note that, if there is only one appearance of the edge between parent and grandparent vertices, a vertex must be added to $M$. Therefore, there are at most $|\mathcal{E}|$ iterations of the inner while loop. Computing $R_u$ is achievable in linear time when $\mathcal{G}_{\downarrow}$ is a tree. Thus, each iteration can be completed in time linear in the number of time-edges. Hence, the algorithm terminates in time $O(|\mathcal{E}|^2)$. □

We let $\mathcal{T}_v$ denote the subtree rooted at a vertex $v$.

**Lemma 3.19.** For any vertex $v$ added to the set $S$ by the algorithm `TaRDiS on Trees`, $S \cap \mathcal{T}_v$ is a TaRDiS of the subtree $\mathcal{T}_v$ rooted at $v$.

*Proof.* We show that all descendants of a vertex $v$ added to $S$ are temporal reachability dominated by $S \cap \mathcal{T}_v$. For $v$ to be added to $S$, it must be either the only vertex adjacent to all vertices in $\mathcal{G} \setminus M$ (and added in line 6), or the parent of a vertex $l$ which is an unmarked vertex at maximum depth (added in line 13). Therefore, all vertices deeper in the subtree than $l$ must be in $M$ immediately prior to the addition of $v$. Note that the children of $v$ are necessarily temporal reachability dominated by $S$ since they are adjacent to the vertex $v$. Therefore, all vertices in $\mathcal{T}_v$ are either temporal reachability dominated by $v$ or in $M$ before $v$ is added to $S$.

We will now show that all vertices in $M \cap \mathcal{T}_v$ before $v$ is added to $S$ are temporally reachable from a vertex in $S \cap \mathcal{T}_v$. We claim that, if a vertex $u$ is in $M \cap \mathcal{T}_v$ but not temporally reachable from $(S \cap \mathcal{T}_v) \setminus \{v\}$, then there is a temporal path from $v$ to $u$. Note that $u$ (and its siblings) must be added to $M$ in line 18, else $u$ would have been added in line 15 and therefore be reachable from $S \setminus \{v\}$. At this point, $u$ must be a vertex not in $M$ of maximum depth. Since $u$ is added to $M$ by line 18, any edge from the parent $p_u$ of $u$ to a child of $p_u$ must be active strictly later (or possibly at the same time in the nonstrict case) than the last appearance of the edge from $p_u$ to the grandparent vertex $g_u$. This implies that any path reaching $p_u$ which traverses $g_u$ must can be made into a temporal path reaching $u$ by appending the edge $(p_u, u)$. Also note that $p_u$ is not in $M$ directly following the addition of $u$ to $M$.

Since we have assumed that $u$ is not a neighbour of $v$ and so $p_u \neq v$, $p_u$ can only be added to $M$ by being temporally reachable from a vertex in $S$ or in line 18. If the former is the case, the path from $v_p$ to $p_u$ must traverse $g_u$ to reach $p_u$. This follows from the fact that $u$ was an unmarked vertex of maximum depth before its addition to $M$ and that all siblings of $u$ are added to $M$ at the same time. Therefore, $v_p$ must be either an ancestor of $u$ or a vertex of depth at most the depth of $u$ such that the path from $v_p$ to $p_u$ traverses an ancestor of $p_u$. Since $\lambda_{\max}(u, p_u) \geq \lambda_{\max}(p_u, g_u) + s$, $u$ must be reachable from $v_p \in S$ by appending the edge $(u, p_u)$ to the aforementioned path.

Now suppose that $p_u$ is added to $M$ by line 18. As before, this gives us that $g_u$ is not in $M$ directly following the addition of $p_u$ to $M$, $p_u$ must be an unmarked vertex of maximum depth previous to its addition to $M$, and any edge from $g_u$ to a child of $g_u$ must be active strictly later (or possibly at the same time in the nonstrict case) than the last appearance of the edge from $g_u$ to its parent. Again, this implies that any temporal path from a vertex in $S$ to $g_u$ which traverses its parent can be extended to give a temporal path from $S$ to $p_u$ and $u$.

The same logic can be applied recursively: if $g_u$ is added to $M$ by line 18, we recurse to its parent, and if $g_u$ is added to $M$ because it is reached by some vertex in $S$ then this vertex reaches $g_u$, and its descendants down to $u$. We must eventually find an ancestor $a_u$ of $u$, reachable from some $v_a$ which is added to $S$, such that the temporal path from $v_a$ to $a_u$ can be extended to reach $u$. We know we must find such a vertex because $v$ is the root of $\mathcal{T}_v$ and, if $u$ and all of its ancestors up to a child of $v$ are marked $M$ by line 18, there is a temporal path from $v$ to $u$. Since $u$ was an arbitrary vertex, we can conclude that all vertices in $\mathcal{T}_v$ must be temporal reachability dominated by a vertex in $S \cap \mathcal{T}_v$. Thus $S \cap \mathcal{T}_v$ is a TaRDiS for $\mathcal{T}_v$. □

**Lemma 3.20.** For any vertex $v$ added to the set $S$ by the algorithm `TaRDiS on Trees`, $S \cap \mathcal{T}_v$ is a minimum TaRDiS of the subtree $\mathcal{T}_v$ rooted at $v$.

*Proof.* We begin by asserting that the TaRDiS $S$ output by Algorithm 1 is a minimal TaRDiS. That is, removing any vertex from $S$ results in at least one vertex which is not temporal reachability dominated by $S$. We show this by contradiction. Assume that $S \setminus \{v\}$ is a TaRDiS for some $v \in S$. Then, immediately before the addition of $v$ to $S$, there is a child $b$ of $v$ which is not in $M$ such that $\lambda\text{max}(b, v) < \lambda_{\min}(v, p) + s$, where $p$ is the parent of $v$. Since the algorithm works from leaves to the root, $b$ cannot be temporally reachable from any vertices in both $S \setminus \{v\}$ and the subtree $\mathcal{T}_v$ rooted at $v$. Else, $b$ would be in $M$ when $v$ is added to $S$ by the algorithm. Since $\lambda\text{max}(b, v) < \lambda_{\min}(v, p) + s$, the vertex $b$ also cannot be temporally reachable from a vertex in $\mathcal{G} \setminus \mathcal{T}_v$. Therefore, $b$ cannot be temporally reachable from any vertex in $S \setminus \{v\}$, and $S$ is a minimal TaRDiS.

Now suppose, for contradiction that such a vertex $v \in S$ and set $S'$ exist where $S' \subseteq T_v$ is a TaRDiS of $\mathcal{T}_v$ and strictly smaller than $S \cap \mathcal{T}_v$. We assume without loss of generality that $S'$ is a minimal TaRDiS.

Consider the set $S' \cap S$. If this is non-empty, let $R$ be the union of reachability sets of vertices in $S' \cap S$. Let $\mathcal{T}'$ be the subgraph of $\mathcal{T}_v$ induced by removing all vertices in $R \setminus (S' \cup S)$. Since we have assumed that $|S'| < |S|$, there must be a connected component $\mathcal{T}''$ of $\mathcal{T}'$ where $|\mathcal{T}'' \cap S'| < |\mathcal{T}'' \cap S|$.

Let $u \in \mathcal{T}'' \cap S$ be the vertex in $(\mathcal{T}'' \cap S) \setminus S'$ of maximum depth. Observe that $u$ is not a leaf in $\mathcal{T}''$, otherwise it would not have an unmarked child before its addition to $S$ by the algorithm, and could not be added to $S$ by line 13. Denote by $c$ the child of $u$ which minimises $\lambda_{\max}(u, c)$. Since $u$ is added to $S$, we must have that $\lambda_{\max}(c, u) < \lambda_{\min}(u, p) + s$, where $p$ is the parent of $u$, when $u$ is added to $S$. Therefore, the path consisting of time-edges $((u, p), \lambda_{\min}(u, p)), ((c, u), \lambda_{\max}(c, u))$ is not a valid temporal path. From here, our goal is to show that any vertex in $S'$ and not $S$ can be replaced with a vertex in $S$ without changing the set of vertices reachable from $S'$. We now have two cases to consider: the case where appearances of the edge $(c, u)$ have been deleted by line 19 of the algorithm, and the case where no appearances of the edge $(c, u)$ have been deleted.

If no appearances of $(u, c)$ have been removed by the algorithm, then $c$ cannot be temporally reachable from $p$ or any ancestors of $p$. Since we know that $u$ is in $S$ and not $S'$, $c$ must be temporally reachable from a descendant $c'$ of $u$ in $S' \setminus S$. We claim that we can swap $c'$ with $u$ without reducing the reachability set of vertices in $S'$. Suppose otherwise; that there is a vertex $w$ no longer reached by $S'$ following the swap of $c'$ and $u$. Since we assumed that $u$ was a vertex of maximum depth in $S' \setminus S$, this contradicts that $S \cap \mathcal{T}_u$ is a TaRDiS of $\mathcal{T}_u$. Thus, this contradicts Lemma 3.19, and we can swap $c'$ with $u$ in $S'$ without reducing the set of vertices temporally reachable from $S'$.

Now suppose that there are appearances of the edge $(u, c)$ which have been removed by the algorithm in line 19. Then, a child of $c$ must have been an unmarked vertex of maximum depth before the deletion of appearances of $(u, c)$. Call this vertex $b$. This implies that $b \in \mathcal{T}''$ since $b$ cannot be temporally reachable from any vertices in $(\mathcal{T}_u \cap S) \setminus \{u\}$. If the algorithm were to add $c$ to $S$ in line 13 when $b$ is unmarked, then $c$ would not be an unmarked vertex of maximum depth in any later iterations of the while loop beginning in line 4. Before deletion of appearances of $(u, c)$ in line 19, we must have that $\lambda_{\max}(b, c) < \lambda_{\max}(c, u) + s$ and $\lambda_{\max}(b, c) \geq \lambda_{\min}(c, u) + s$. This implies that there can be no temporal path from $p$ or an ancestor of $p$ to $b$ using the removed time-edge $((c, u), \lambda_{\max}(c, u))$. Following deletion of the appearance $\lambda_{\max}(c, u)$ (potentially multiple times) in line 19, $\lambda_{\max}(b, c) \geq \lambda_{\max}(c, u) + s$ and $\lambda_{\max}(c, u) < \lambda_{\min}(u, p) + s$. This follows from the fact that $u$ is added to $S$ in line 13. Therefore, there is no temporal path from $p$ or an ancestor of $p$ to $b$ following the deletion. Since we know that $u$ is in $S$ and not $S'$, $b$ must be temporally reachable from a descendant $b'$ of $u$ in $S' \setminus S$. We claim that we can swap $b'$ with $u$ without reducing the reachability set of vertices in $S'$. Suppose otherwise; that there is a vertex $w$ no longer reached by $S'$. Since we assumed that $u$ was a vertex of maximum depth in $S' \setminus S$, this contradicts that $S \cap \mathcal{T}_u$ is a TaRDiS of $\mathcal{T}_u$. Thus, this also contradicts Lemma 3.19.

In both cases, we have shown that there is a vertex in $S' \setminus S$ that can be replaced by $u$ without reducing the reachability set of vertices in $S'$. We repeat this with the vertex at maximum depth in $\mathcal{T}'' \cap S \setminus S'$ until either we contradict the assumption that $S'$ is a TaRDiS of $\mathcal{T}_v$, or obtain $S = S'$; contradicting our assertion that $|S'| < |S|$. Therefore $S$ is a minimum TaRDiS of $\mathcal{T}_v$. □

**Lemma 3.21.** The set $S$ output by the algorithm `TaRDiS on Trees` is a minimum TaRDiS of $\mathcal{G}$.

*Proof.* If the root $r$ of $\mathcal{G}$ is in $S$ as output by the algorithm, then combining Lemmas 3.19 and 3.20 gives us the desired result. Suppose $r$ is not in $S$. Then, rooting the tree at the final vertex added to $S$ gives the result without changing the vertices in $S$. □

This gives us the following theorem and corollary.

**Theorem 3.22.** When the footprint of the graph is a tree, TaRDiS is solvable in $O(|\mathcal{E}|^2)$ time.

We obtain a running time dependent only on the number of vertices for simple temporal graphs, where $|\mathcal{E}| = |E(\mathcal{G}_\downarrow)| = V(\mathcal{G}) - 1$.

**Corollary 3.23.** When the input temporal graph is simple and the footprint of the graph is a tree, TaRDiS is solvable in $O(n^2)$ time.

In Section 3.4, we give a more general algorithm which solves any variant of TaRDiS on a nice tree decomposition of the underlying graph. When the footprint graph is a tree, we have treewidth 1. Our algorithm for TaRDiS on trees runs faster than the tree decomposition algorithm (whose runtime also grows with $\tau$), so this does not subsume Theorem 3.22.

## 3.3 Classical complexity results for MaxMinTaRDiS

We now shift our focus to MaxMinTaRDiS, where the objective is to find a temporal assignment which precludes the existence of a small TaRDiS. For each variant, we characterize the minimum lifetime such that the problem becomes intractable. A leap in complexity to $\Sigma_2^P$ is expected because the problem implicitly quantifies twice: "Does there *exist* a temporal assignment such that *for all* sets of vertices of cardinality at most $k-1$, the set is not a TaRDiS", but intriguingly this is only seen with Happy MaxMinTaRDiS (though we do not exclude it for Nonstrict MaxMinTaRDiS).

**Definition 3.24** ($\Sigma_2^P$ - adapted from [113], Definition 5.1 and Theorem 5.15)**.** $\Sigma_2^P$ is the set of all languages $L$ for which there exists a polynomial-time Turing Machine $M$ and a polynomial $q$ such that

$$x \in L \iff \exists u \in \{0,1\}^{q(|x|)} \forall v \in \{0,1\}^{q(|x|)} M(x,u,v) = 1$$

for every $x \in \{0,1\}^*$.

Equivalently, $\Sigma_2^P$ is the set of all languages $L$ which can be recognized by a nondeterministic polynomial Turing Machine with access to an oracle for an NP-complete language.[5]

Arora and Barak [113] note that NP $\cup$ co-NP $\subseteq \Sigma_2^P$ and that $\Sigma_1^P =$ NP. The complement of a $\Pi_i^P$-complete problem is necessarily $\Sigma_i^P$-complete [114]. It is widely believed that $\Sigma_2^P$ is a strict superclass of NP $\cup$ coNP. In this sense, Happy MaxMinTaRDiS is a harder problem than both Strict MaxMinTaRDiS (which is coNP-complete), and all TaRDiS variants (which are NP-complete as shown in Section 3.2). We show Nonstrict MaxMinTaRDiS is at least as hard as all TaRDiS variants and at most as hard as Happy MaxMinTaRDiS, but leave open its exact complexity. Interestingly, this is achieved by proving that the well-studied Distance-3 Independent Set problem is a subproblem of Nonstrict MaxMinTaRDiS.

### 3.3.1 Containment in $\Sigma_2^P$, useful tools, and small lifetime

Here we give some preliminary complexity results for each variant of MaxMinTaRDiS. We begin by showing that they are all contained in $\Sigma_2^P$. We then show that we need only consider simple temporal assignments when trying to solve MaxMinTaRDiS. Finally, we draw comparisons between Happy MaxMinTaRDiS and Strict MaxMinTaRDiS and the classical problems Edge Colouring and Dominating Set respectively, which allow us to show NP-hardness and coNP-hardness of the respective problems.

**Lemma 3.25.** Each variant of the problem MaxMinTaRDiS is contained in $\Sigma_2^P$.

[5]Classically, the NP-complete language used for the definition is Satisfiability, but we shall later use TaRDiS for convenience.

*Proof.* A nondeterministic polynomial Turing Machine with access to an oracle for (Strict/Nonstrict/Happy) TaRDiS can decide whether an instance $(H, k, \tau)$ is a yes-instance of (Strict/Nonstrict/Happy) MaxMinTaRDiS by first nondeterministically guessing a temporal assignment $\lambda : E(H) \to [\tau]$ and then accepting if and only if $(H, \tau, k-1)$ is a no-instance of (the relevant variant of) TaRDiS, by using the oracle. $\square$

We begin with the observation that we need only consider *simple* temporal assignments for the input graph.

**Lemma 3.26.** Let $(H, k, \tau)$ be a yes-instance of MaxMinTaRDiS. Then there exists a *simple* temporal assignment $\lambda : E \to [\tau]$ such that the cardinality of the minimum TaRDiS on $(H, \lambda)$ is at least $k$.

*Proof.* We begin by supposing, for a contradiction, that $(H, k, \tau)$ is an instance of MaxMinTaRDiS such that any optimal solution is non-simple. Denote such a solution by $\lambda^*$. By our assumption, there is at least one edge $e^* \in E(H)$ such that $|\lambda^*(e^*)| > 1$. Let $\lambda$ be a simple temporal assignment such that, for all edges $e \in E(H)$, $\lambda(e) \in \lambda^*(e)$. Then, under $\lambda$, the reachability set of any vertex $v \in V(H)$ is a subset of the reachability set of $v$ under $\lambda^*$. Therefore, a minimal TaRDiS of $(H, \lambda)$ must be at most the cardinality of a minimal TaRDiS of $(H, \lambda^*)$. Since $\lambda$ is a simple temporal assignment, we have contradicted our assumption. Thus, for every instance of MaxMinTaRDiS, there exists an optimal solution which is simple. $\square$

**Lemma 3.27** ([78])**.** A static graph $H$ admits a happy temporal assignment $\lambda : E(G) \to [\tau]$ if and only if $H$ is $\tau$-edge colourable.

*Proof.* To see this, we assign each of the times a colour. Then, if the graph cannot be $\tau$-edge coloured, we cannot assign times to the edges such that no two adjacent edges share a time. Furthermore, if we can give $H$ a happy temporal assignment, then the corresponding assignment of colours to edges is a proper edge-colouring. $\square$

Happy MaxMinTaRDiS restricted to instances with $k = 0$ asks only if there exists a happy assignment $\lambda$ with lifetime $\tau$ for the input graph $G$. This is equivalent to the Edge colouring problem with $\tau$ colours. Edge Colouring is NP-complete, even when the number of colours is 3 [115].

**Corollary 3.28.** Happy MaxMinTaRDiS is NP-hard for any $\tau \geq 3$, even when $k = 0$.

**Lemma 3.29.** Nonstrict MaxMinTaRDiS can be computed in linear time when $\tau = 1$. When $\tau \leq 2$, Happy MaxMinTaRDiS is solvable in linear time.

*Proof.* If every edge is active at the same time, we have no choice in the temporal assignment. As shown in Lemma 3.8, we can find the size of the minimum nonstrict TaRDiS in linear time by counting the number of connected components.

As stated in Lemma 3.8, the footprint of a happy temporal graph with lifetime 2 necessarily consists of disconnected paths and even cycles. In paths and cycles of even length, there is only one happy temporal assignment up to symmetry. In odd paths, the optimal ordering is that wherein both edges incident to leaves are assigned time 1. This forces one of the endpoints of these edges to be in a TaRDiS, which is not the case if they are assigned time 2. Having found a single labelling $\lambda$ to consider, we may now return YES if and only if $(G, \lambda, k-1)$ is a no-instance of HAPPY TARDIS (which may be decided in linear time applying Lemma 3.8). □

**Lemma 3.30.** For any static graph $H$, $k \in \mathbb{N}^+$ and $\tau \in \mathbb{N}^+$, $(H, k, \tau)$ is a yes-instance of STRICT MAXMINTARDIS if and only if $(H, k-1)$ is a no-instance of DOMINATING SET.

*Proof.* We first show that the constant function $\lambda$ is an optimal one (i.e. one which maximizes the size of the minimum TaRDiS for $(H, \lambda)$). Suppose otherwise, that we have an optimal temporal assignment $\lambda'$ where there exist edges $e$, $e'$ such that $\lambda'(e) \neq \lambda'(e')$. Then, under a temporal assignment $\lambda(e) = c$ for all edges $e \in E$ and $c \in \mathbb{N}$, all vertices have reachability sets whose cardinality are bounded above by the size of their reachability set under $\lambda'$. Therefore a minimum TaRDiS under $\lambda$ must be at least the size of a minimum TaRDiS under $\lambda'$. Applying Lemma 3.6, we see that $(H, k)$ is a yes-instance of STRICT MAXMINTARDIS if and only if $(H, k-1)$ is a no-instance of DOMINATING SET. □

**Corollary 3.31.** STRICT MAXMINTARDIS is coNP-complete, coW[2]-complete with respect to $k$, and para-NP-hard with respect to $\Delta + \tau$.

### 3.3.2 $\Sigma_2^P$-completeness of HAPPY MAXMINTARDIS with lifetime 3

As stated in Corollary 3.28, HAPPY MAXMINTARDIS is trivially NP-hard even for lifetime $\tau = 3$. Lemma 3.25 gives us that the problem is in $\Sigma_2^P$. We characterize the problem's complexity exactly, showing it is $\Sigma_2^P$-complete.

We begin by presenting the problem RESTRICTED PLANAR SATISFIABILITY, which is $\Pi_2^P$ complete [116].

RESTRICTED PLANAR SATISFIABILITY (RPS)
*Input:* An expression of form $(\forall X)(\exists Y)\Phi(X, Y)$ with $\Phi$ a CNF formula over the set $X \cup Y$ of variables. Each clause contains exactly 3 distinct variables, each variable occurs in exactly three clauses, every literal appears at most twice, and

the graph $G_\Phi$ is planar.

*Question:* Is the expression true?

We also introduce the complement problem co-RPS, which we will reduce from.

co-Restricted Planar Satisfiability (co-RPS)

*Input:* An expression of form $(\exists X)(\nexists Y)\Phi(X, Y)$ with $\Phi$ a CNF formula over the set $X \cup Y$ of variables. Each clause contains exactly 3 distinct variables, each variable occurs in exactly three clauses, every literal appears at most twice, and the graph $G_\Phi$ is planar.

*Question:* Is the expression true?

**Lemma 3.32.** The problem co-RPS is $\Sigma_2^P$ complete.

*Proof.* Gutner shows Restricted Planar Satisfiability (RPS) to be $\Pi_2^P$-complete in [116]. Note that they do not state the restriction that each literal appears at most twice in the lemma stating their result, but this can be seen from their construction. □

### Intuition

We can imagine problems in $\Sigma_2^P$ as games with two players. In co-RPS the first player chooses an assignment to variables in $X$, then the second player chooses an assignment to the variables in $Y$. If the resulting assignment to $X \cup Y$ satisfies $\Phi$ then the second player wins; otherwise the first player wins. Analogously, in Happy MaxMinTaRDiS the first player chooses a happy temporal assignment $\lambda$ with lifetime $\tau$ for the edges of $G$, then the second player chooses a set $S$ of vertices in $G$ of size at most $k - 1$. If $S$ is a TaRDiS of $(G, \lambda)$ then the second player wins; otherwise the first player wins.

We perform this reduction by creating gadgets which force the first player to choose a function $\lambda$ which is *nice*. Essentially, $\lambda$ must assign specific times to certain edges. This means that if the first player does not do this, the second player can always win. We can then replicate some techniques from the proof of Theorem 3.12 to encode clauses being satisfied. The choice of TaRDiS by the second player then encodes a truth assignment to variables in $Y$. By allowing a small amount of freedom in the choice of $\lambda$, we allow the first player to encode a truth assignment to the variables of $X$. We now give the formal construction.

### Construction

Given an instance $(\Phi, X, Y)$ of co-RPS, we will produce an instance $(G, k, \tau = 3)$ of MaxMinTaRDiS. The number of clauses in $\Phi$ is denoted by $m$. We denote the number of variables in $X$ (respectively $Y$) with $n_X$ (resp. $n_Y$). The total number of variables is $n$. Note that $3m \geq n \geq m$, so the size of the co-RPS instance is linear in

$n$. Further, we denote $X = \{x_1, \dots, x_{n_x}\}$ and $Y = \{y_{n_x+1}, \dots, y_{n_x+n_y}\}$. We say that the literal $l_{2i}$ (respectively, $l_{2i-1}$) is the positive (resp., negative) literal for variable $x_i$ if $i \leq n_x$, and the positive (resp. negative) literal for variable $y_i$ if $n_x < i \leq n_x + n_y$.

We now describe the construction of $G$ and $k$. The construction of our gadgets is intended to guarantee that any optimal temporal assignment $\lambda$ will have certain properties. First, we define the Uncovered 2-Gadget, Uncovered 3-Gadget, and Covered 2-Gadget. Each of these is treated in our construction as a vertex of degree one. The construction for each of these makes use of large values $\alpha$ and $\beta$. Let $\beta = 100n$ and $\alpha = 600n\beta + 600n + 2$. In particular we require that $\alpha >> \beta >> n$, $\beta$ is even, and $\alpha \equiv 2 \mod 12$.

### Uncovered 2-Gadget (U2G)

This construction is illustrated in Figure 3.9. We use a U2G by connecting it to some vertex $x$ elsewhere in the construction. The intuition is that a U2G will force the edge connecting it to $x$ to be assigned time 2, and moreover that a "good" choice of TaRDiS inside the U2G will not reach (cover) $x$. Given some such $x$, we create vertices $y$ and $w$ and a ladder graph on $2\alpha$ vertices $\{u_1, \dots, u_\alpha, v_1, \dots, v_\alpha\}$, with the edges $(x, y), (y, u_1), (y, v_1), (w, u_\alpha), (w, v_\alpha)$. The *ladder graph* $L_n$ on $2n$ vertices has vertex set $V(L_n) = \{u_1, \dots, u_n, v_1, \dots, v_n\}$, and edge set $E(L_n) = \{(u_i, u_{i+1}) | i \in [n-1]\} \cup \{(v_i, v_{i+1}) | i \in [n-1]\} \cup \{(u_i, v_i) | i \in [n]\}$.

### Covered 2-Gadget (C2G)

The construction is very similar to that of the U2G - the only difference is the incrementation of the ladder length by 1. Much like a U2G, the C2G will (informally) force the edge connecting it to the external vertex $x$ to be assigned time 2, but now any "good" choice of TaRDiS inside the C2G *will* reach (cover) $x$. Given $x$, we create vertices $y$ and $w$ and a ladder graph on $2\alpha + 2$ vertices $\{u_1, \dots, u_{\alpha+1}, v_1, \dots, v_{\alpha+1}\}$, and edges $(x, y), (y, u_1), (y, v_1), (w, u_{\alpha+1}), (w, v_{\alpha+1})$.

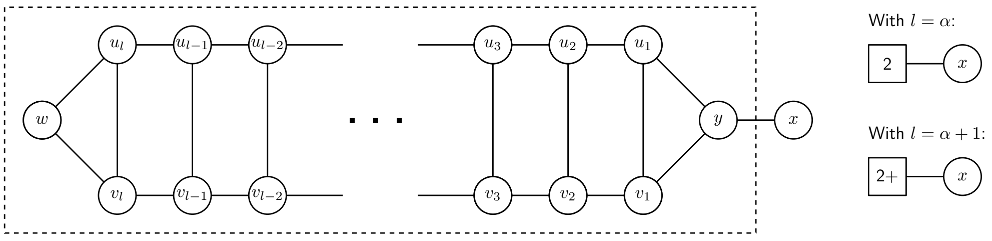


Figure 3.9: A vertex $x$ incident to the Uncovered 2 Gadget (U2G) or Covered 2 Gadget (C2G). Left: construction of the gadget where $l = \alpha$ for a U2G and $l = \alpha + 1$ for a C2G. Right: depiction of the gadget used in later figures. Note that planarity is preserved.

### Uncovered 3-Gadget (U3G)

Informally, this gadget forces the edge connecting it to an external vertex $x$ to be assigned time 3, and does not cover $x$. The construction is illustrated in Figure 3.10.

We start with the ladder graph on $2\alpha$ vertices $\{u_1, \ldots, u_\alpha, v_1, \ldots, v_\alpha\}$, and doubly subdivide the edge $(u_i, u_{i+1})$ (respectively $(v_i, v_{i+1})$) whenever $i$ is even (resp. odd) with two new vertices $a_i, b_i$. We say *doubly subdivide* an edge $(u, v)$ to refer to the deletion of $(u, v)$ and introduction of two vertices $a, b$ and three edges $(u, a), (a, b), (b, v)$. We add the vertices $y_1, y_2, y_3, w$ and edges $(x, y_1)$, $(y_1, y_2)$, $(y_2, y_3)$, $(y_3, u_1)$, $(y_3, v_1)$, $(w, u_\alpha)$. We denote the set of internal vertices of the U3G by $V_{\text{U3G}} = \{a_1, \ldots, a_{\beta-1}, b_1, \ldots b_{\beta-1}, y_1, y_2, y_3, u_1, \ldots, u_\beta, v_1, \ldots, v_\beta, w\}$. Internal vertices are highlighted by a red (solid) box in Figure 3.10. We make any vertex not already of degree 3 in $V_{\text{U3G}}$ incident to a U2G.

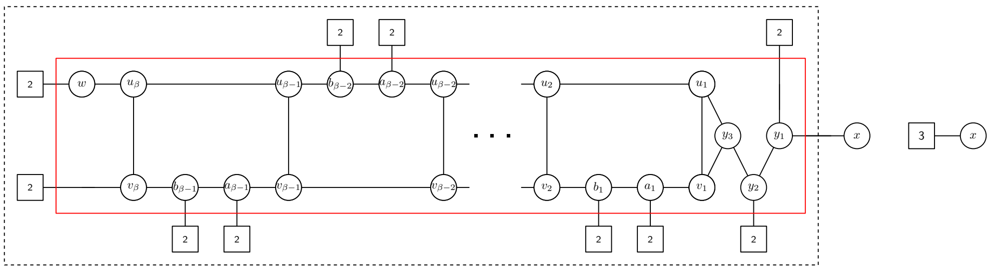


Figure 3.10: A vertex $x$ incident to the Uncovered 3 Gadget (U3G). Left: construction of the gadget; vertices of $V_{\text{U3G}}$ are in the red (solid) box. Right: depiction of the gadget used in later figures. Note that planarity is preserved.

### Uncovered 1-Gadget (U1G)

Informally, this gadget forces the edge connecting it to an external vertex $x$ to be assigned time 1, and does not cover $x$. For a U1G, we simply create a vertex $y$ incident to both a C2G and a U3G as shown in Figure 3.11, then add an edge from $y$ to the target vertex $x$.

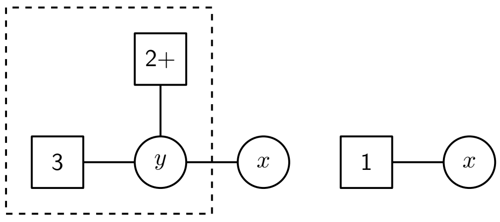


Figure 3.11: A vertex $x$ incident to the Uncovered 1 Gadget (U1G). Left: implementation of the gadget. Right: depiction of the gadget used in later figures.

### Construction: literal vertices and clause gadgets

Exactly $3m$ *literal vertices* and $m$ *clause gadgets* are created. We iterate over $\Phi$, and for each clause $c_j$ we create a cycle on 6 new vertices $Q_j = \{q_j^1, \ldots, q_j^6\}$, and make $q_j^2, q_j^4, q_j^6$ each incident to a U2G. Where the clause $c_j$ contains the $a$th appearance of some literal $l_i$ in $\Phi$, we create two new *literal vertices* $l_i^a$ and $\overline{l_i^a}$ connected by an edge, and make each of $l_i^a$ and $\overline{l_i^a}$ incident to a new U1G. We make $l_i^a$ incident to a vertex from $\{q_j^1, q_j^3, q_j^5\}$, such that each of these is adjacent to exactly one literal vertex. Figure 3.12 illustrates this construction. Vertices $u, v, w, \overline{u}, \overline{v}, \overline{w}$ are literal vertices for which the corresponding literals appear in $c_j$. The intention is that only *nice* assignments (like the one shown in Figure 3.12) to the clause gadget need to be considered, and that under a nice assignment any canonical TaRDiS of small size will include exactly two vertices belonging to the clause gadget.

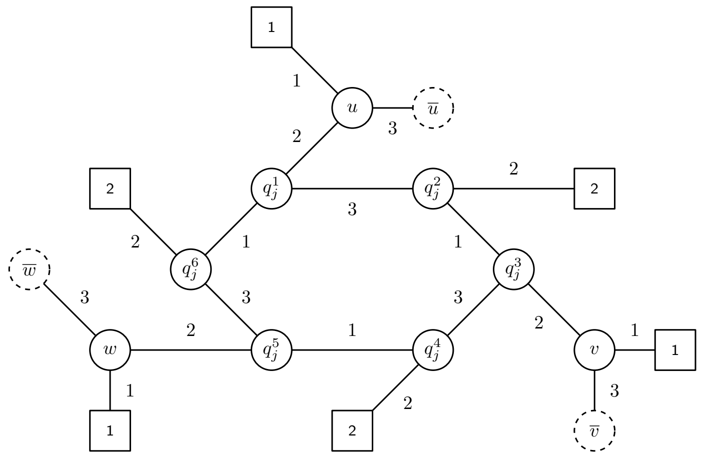


Figure 3.12: The clause gadget for clause $c_j$ together with an example *nice* assignment. If, for example, clause $c_j = (l_5 \vee l_{17} \vee l_{20})$ is the first appearance of $l_5$ and $l_{20}$ and the second appearance of $l_{17}$ in $\Phi$, then $u = l_5^1$, $v = l_{17}^2$ and $w = l_{20}^1$. The neighbourhoods of dashed vertices are shown in Figures 3.13 and 3.14 respectively depending on whether they correspond to variables in $X$ or $Y$.

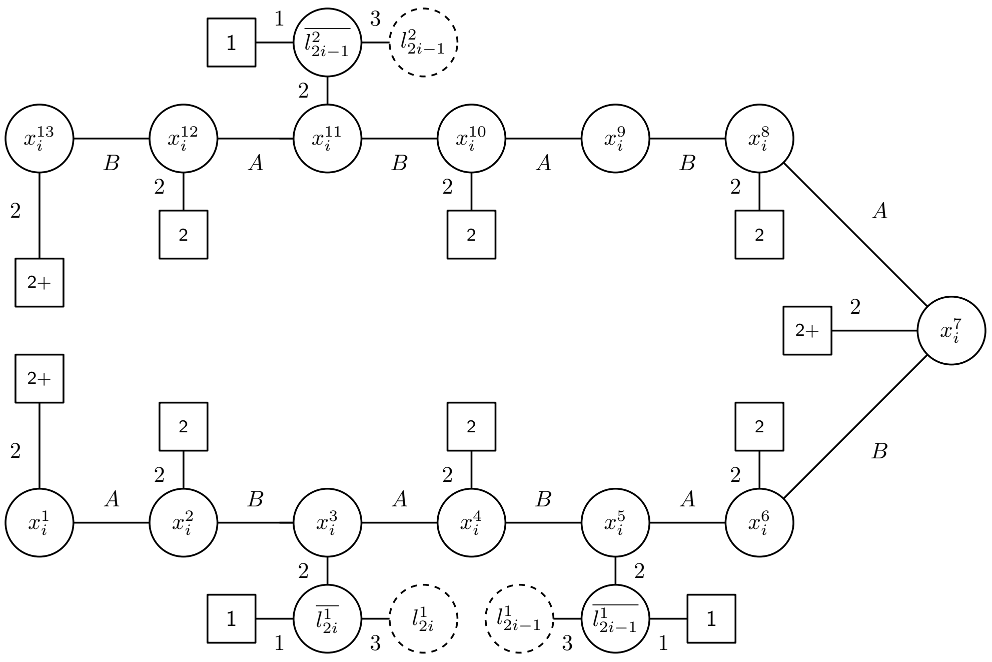


Figure 3.13: The $X$-variable gadget for variable $x_i$ together with an example gadget-respecting assignment; note that either $A = 1$ and $B = 3$ or $B = 1$ and $A = 3$. Only in the former case, which corresponds to setting $x_i$ to `True`, does $x_i^4$ have a temporal path to $l_{2i}^1$. Here $x_i$ appears twice negatively and once positively, hence $x_i^9$ is not incident to a literal vertex. The neighbourhoods of dashed vertices are shown in Figure 3.12.

## Construction: $X$-variable gadget

The intuition here is that there is a little freedom in the choice of temporal assignment locally, and that this choice will encode an assignment to the corresponding variable – this is illustrated in Figure 3.13. For each variable $x_i \in X$, we create 13 vertices $\{x_i^1, \ldots, x_i^{13}\}$ connected in a path, with vertex $x_i^p$ incident to a U2G if $p$ is even, and incident to a C2G for $p \in \{1, 7, 13\}$. Lastly we add the edges $(x_i^3, \overline{l_{2i}^1})$, $(x_i^5, \overline{l_{2i-1}^1})$, $(x_i^9, \overline{l_{2i}^2})$,

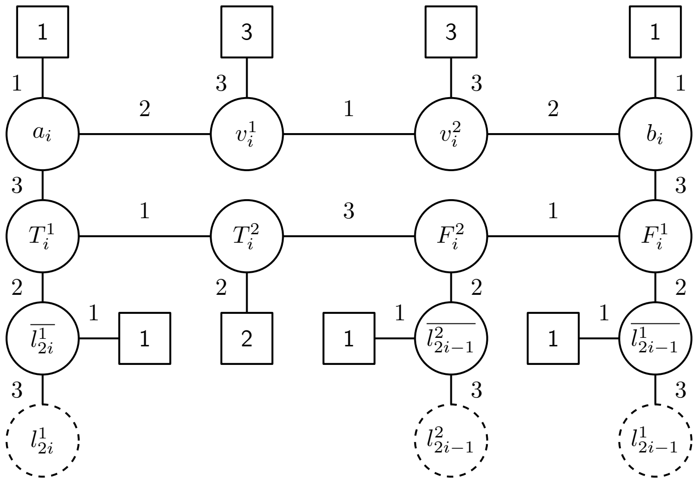


Figure 3.14: The $Y$-variable gadget for variable $y_i$ together with an example *nice* assignment. Note the similarity to the variable gadget in Figure 3.6. Here $x_i$ appears twice negatively and once positively, hence $T_i^2$ is connected to a U2G and not a literal vertex. The neighbourhoods of dashed vertices are shown in Figure 3.12.

and $(x_i^{11}, \overline{l_{2i-1}^2})$ whenever the appropriate literal vertex exists. Recall that, by the restrictions on $\Phi$, at most one of the vertices $l_{2i}^2$ and $l_{2i-1}^2$ exist in our construction for any input.

**Construction: $Y$-variable gadget**

Conversely, the $Y$-variable gadget leaves no (meaningful) freedom in the temporal assignment, but does leave some freedom in the choice of vertices to include in a TaRDiS – again encoding a Boolean assignment to the variable. This construction is illustrated in Figure 3.14. For each variable $y_i \in Y$, we create 8 vertices $\{a_i, v_i^1, v_i^2, b_i, F_i^1, F_i^2, T_i^2, T_i^1\}$ connected in a cycle, with vertices $v_i^1$ and $v_i^2$ incident to U3Gs and vertices $a_i$ and $b_i$ incident to U1Gs. Then we add edges $(T_i^1, \overline{l_{2i}^1}), (T_i^2, \overline{l_{2i}^2})$, $(F_i^2, \overline{l_{2i-1}^2}), (F_i^1, \overline{l_{2i-1}^1})$ whenever the appropriate literal vertex exists. If vertex $T_i^2$ is not incident to a literal vertex, we connect it to a U2G. Otherwise, $F_i^2$ is connected to a U2G.

**Construction: $k$**

In order to state $k$, we first define some auxiliary variables. As stated earlier, the number of $X$-gadgets, $Y$-gadgets and clause-gadgets is $n_X, n_Y$ and $m$ respectively. We then define:

$$\begin{aligned}
\#_{\text{lit}} &= 2 \cdot 3m && \text{(The number of literal nodes.)}\\
\#_{\text{U1G}} &= \#_{\text{lit}} + 2 \cdot n_Y && \text{(The number of U1Gs.)}\\
\#_{\text{U3G}} &= 2n_Y + \#_{\text{U1G}} && \text{(The number of U3Gs.)}\\
\#_{\text{C2G}} &= 3n_X + \#_{\text{U1G}} && \text{(The number of C2Gs.)}\\
\#_{\text{U2G}} &= \#_{\text{U3G}}(2\beta + 2) + 3m + 6n_X + n_Y && \text{(The number of U2Gs.)}
\end{aligned}$$

We now define $k$:

$$k = \#_{\text{U2G}} \cdot \left(\frac{\alpha+1}{3}\right) + \#_{\text{C2G}} \cdot \left(\frac{\alpha+4}{3}\right) + \#_{\text{U3G}} \cdot (\beta) + 2m + 3n_X + 3n_Y + 1$$

This concludes the construction of the Happy MaxMinTaRDiS instance.

## Properties of the construction

We first define the temporal assignments of interest to us.

**Definition 3.33** (Gadget-respecting, Nice Temporal Assignment)**.** A happy temporal assignment $\lambda$ for the graph $G$ described above is:

- *2-gadget-respecting* if every edge $(x, y)$ incident to a U2G or C2G is assigned time 2 under $\lambda$.
- *3-gadget-respecting* (resp. *1-gadget-respecting*) if every edge $(x, y)$ incident to a U3G (resp. U1G) is assigned time 3 (resp. 1) under $\lambda$.

If $\lambda$ is 1-, 2- and 3-gadget-respecting, we say that $\lambda$ is *nice*.

We will show that a happy temporal assignment $\lambda$ for $G$ satisfies that every TaRDiS of $(G, \lambda)$ has cardinality at least $k - 1$ if and only if $\lambda$ is *nice*. We then show that there exists a $\lambda$ such that every TaRDiS of $(G, \lambda)$ has cardinality at least $k$ if and only if $(\Phi, X, Y)$ is a yes-instance of co-RPS.

**Lemma 3.34.** For any happy temporal assignment $\lambda$ on any U2G gadget incident to $x$ consisting of vertices $V_{\text{U2G}} = \{y, w, u_1, \ldots, u_\alpha, v_1, \ldots, v_\alpha\}$, if $\lambda(x, y) = 2$

- exactly $\frac{\alpha+1}{3}$ vertices from $V_{\text{U2G}}$ are needed to temporal reachability dominate $V_{\text{U2G}}$;
- no choice of $\frac{\alpha+1}{3}$ vertices temporally dominates $V_{\text{U2G}} \cup \{x\}$.

If $\lambda(x, y) \neq 2$, it is possible to temporally dominate $V_{\text{U2G}}$ with at most $\frac{\alpha+2}{4}$ vertices.

*Proof.* It is easy to check that there is only one proper 3-colouring of the edges of the induced graph on $V_{\text{U2G}} \cup \{x\}$ up to isomorphism. Denote by $A$ the colour of $(x, y)$, $B$ the colour of $(y, u_1)$ and $C$ the colour of $(y, v_1)$. Note that all edges $(u_i, v_i)$ are given colour $A$, and the edges on the path $u_1, \ldots, u_\alpha$ (resp. $v_1, \ldots, v_\alpha$) alternate between $B$ and $C$.

Any possible happy temporal assignment then corresponds exactly to one of 6 possible assignments of times $\{1, 2, 3\}$ to the colours $\{A, B, C\}$. From this point, we abuse notation slightly and denote $\lambda(x, y)$ by $\lambda(A)$, $\lambda(y, u_1)$ by $\lambda(B)$ and $\lambda(y, v_1)$ by $\lambda(C)$. By symmetry of the gadget construction, we can always assume $\lambda(B) < \lambda(C)$.

If $\lambda(A) = 2$, we have $\lambda(B) = 1$ and $\lambda(C) = 3$; then $|R_v| \leq 6$ for all $v \in V_{\text{U2G}}$. Note in particular that $R_{v_1} = \{y, v_1, v_2, v_3, u_1, u_2\}$, and $R_x \cap V_{\text{U2G}} = \{y, v_1\}$. Thus for any TaRDiS $S$, at least $|V_{\text{U2G}}| - 2 = 2\alpha$ vertices in $V_{\text{U2G}}$ are reached by vertices of $S \cap V_{\text{U2G}}$, each of which reaches at most 6 vertices including itself. Recall that $\alpha \equiv 2 \mod 12$. Hence, $|S \cap V_{\text{U2G}}| \geq \lceil \frac{2\alpha}{6} \rceil = \frac{\alpha+1}{3}$. Note that, by the same logic, any set of size at most $\frac{\alpha+1}{3}$ reaches at most $2\alpha+2$ vertices and hence no such set reaches every vertex in $V_{\text{U2G}} \cup \{x\}$ since $|V_{\text{U2G}} \cup \{x\}| = 2\alpha + 3$. Further, the set $\{v_1, u_4, v_7, u_{10}, \dots v_{\alpha-1}\}$ is of size $\frac{\alpha+1}{3}$ exactly and reaches every vertex in $V_{\text{U2G}}$. Conversely, if $\lambda(A) \in \{1, 3\}$ then $\{v_1, v_5, v_9, \dots, v_{\alpha-1}\}$ is a set of size $\frac{\alpha+2}{4}$ which reaches all vertices of $V_{\text{U2G}}$. □

**Lemma 3.35.** For any happy temporal assignment $\lambda$ on any C2G gadget incident to vertex $x$ consisting of vertices $V_{\text{C2G}} = \{y, w, u_1, \dots, u_{\alpha+1}, v_1, \dots, v_{\alpha+1}\}$, if $\lambda(x, y) = 2$

- exactly $\frac{\alpha+4}{3}$ vertices from $V_{\text{C2G}}$ are needed to temporally dominate $V_{\text{C2G}}$;
- it is possible to temporally dominate $V_{\text{C2G}} \cup \{x\}$ with the same number of vertices.

If $\lambda(x, y) \neq 2$ it is possible to temporally dominate $V_{\text{C2G}}$ with at most $\frac{\alpha+6}{4}$ vertices.

*Proof.* As in the case of the U2G, there is only one proper 3-colouring of the edges of the induced graph on $V_{\text{U3G}} \cup \{x\}$ up to isomorphism. We again denote $A$ the colour of $(x, y)$, $B$ the colour of $(y, u_1)$ and $C$ the colour of $(y, v_1)$. Since gadget symmetry is preserved, we also may assume without loss of generality that $\lambda(B) < \lambda(C)$. Then the assignment of the edges on vertices from $V_{\text{C2G}} \cup \{x\}$ depends only on the value of $\lambda(x, y)$.

The dependence of reachability on this choice remains; namely if $\lambda(A) = 2$ then $|R_v| \leq 6$ for all $v \in V_{\text{C2G}}$ and $R_x \cap V_{\text{C2G}} = \{y, v_1\}$. Applying the same logic as above, for any TaRDiS $S$ at least $|V_{\text{C2G}}| - 2 = 2\alpha + \mathbf{2}$ vertices in $V_{\text{C2G}}$ are reached by vertices of $S \cap V_{\text{C2G}}$. Hence $|S \cap V_{\text{C2G}}| \geq \lceil \frac{2\alpha+2}{6} \rceil = \frac{\alpha+4}{3}$. Now $\{y, u_2, v_5, u_8, v_{11}, \dots u_\alpha\}$ is of size $\frac{\alpha+4}{3}$ exactly and reaches every vertex in $V_{\text{C2G}}$ and additionally reaches $x$ at time 2. Conversely if $\lambda(A) \neq 2$ then covering $V_{\text{C2G}}$ requires at most $\frac{\alpha+6}{4}$ vertices, for example $\{y, u_2, u_6, u_{10}, \dots, u_\alpha\}$. □

**Lemma 3.36.** For any happy function $\lambda$ which is *not* 2-gadget-respecting, $(G, \lambda)$ admits a TaRDiS of size less than $k - 1$.

*Proof.* Recall that $n = n_Y + n_X$ and $n > m$. Observe that the number of vertices in $G$ which are not inside a U2G or C2G is exactly:

$$\begin{aligned}
&\#_{\text{U3G}} \cdot (4\beta + 2) + \#_{\text{U1G}} + \#_{\text{lit}} + 8n_Y + 13n_X + 6m \\
&= (2n_Y + 6m)(4\beta + 2) + 6m + 6m + 12n_Y + 13n_X + 16m \\
&\leq 32n\beta + 31n.
\end{aligned}$$

which is strictly smaller than $\frac{\alpha}{12} \approx 50n\beta + 50n$. Even if there exists some $\lambda_\perp$ such that every vertex in $G$ not belonging to a U2G or C2G is necessarily included in every TaRDiS of $(G, \lambda_\perp)$, any such TaRDiS would still have size less than $\#_{\text{U2G}} \cdot \frac{\alpha+1}{3} + \#_{\text{C2G}} \cdot \frac{\alpha+4}{3}$ and hence less than $k - 1$. □

**Lemma 3.37.** For any 2-gadget respecting happy temporal assignment $\lambda$ on any U3G gadget incident to vertex $x$:

- If $\lambda(x, y_1) = 3$ then $\beta$ vertices from $V_{\text{U3G}}$ are needed to temporally dominate $V_{\text{U3G}} \setminus \{y_1\}$, and no choice of $\beta$ vertices temporally dominates $V_{\text{U3G}} \cup \{x\}$.
- If $\lambda(x, y_1) \neq 3$, it is possible to temporally dominate $V_{\text{U3G}} \setminus \{y_1\}$ with at most $\frac{\beta}{2} + 1$ vertices.

Recall $V_{\text{U3G}} = \{a_1, \ldots, a_{\beta-1}, b_1, \ldots b_{\beta-1}, y_1, y_2, y_3, u_1, \ldots, u_\beta, v_1, \ldots, v_\beta, w\}$; the set of vertices in the red (solid) box in Figure 3.10.

*Proof.* There are only two possible 2-gadget-respecting assignments for the edges of $V_{\text{U3G}}$. In any case, for all $i \in [\beta]$, $\lambda(u_i, v_i) = \lambda(x, y_1)$. We denote by $\lambda_3$ the assignment where $\lambda(x, y) = 3$ and $\lambda_1$ the temporal assignment where $\lambda(x, y) = 1$.

It can be manually verified that, under $\lambda_3$, $\beta$ vertices are necessary to temporally dominate $V_{\text{U3G}} \setminus \{y_1\}$. This is also sufficient: the set $\{v_1, u_2, v_3, u_4, \ldots, u_\beta\}$ is one possibility. On the other hand, under $\lambda_1$ there exists a set of size $\frac{\beta}{2} + 1$ which temporally dominates $V_{\text{U3G}} \setminus \{y_1\}$, namely $\{u_1, u_2, u_4, \ldots, u_\beta\}$. □

**Lemma 3.38.** For any 2-gadget-respecting $\lambda$, if for any edge $e$ incident to a U3G $\lambda(e) \neq 3$ then $(G, \lambda)$ admits a TaRDiS of size less than $k - 1$.

*Proof.* Observe that the number of vertices in $G$ which are not inside a U2G, C2G or U3G is exactly:

$$\begin{aligned} &\#_{\text{U1G}} + \#_{\text{lit}} + 8n_Y + 13n_X + 6m \\ &= 6m + 6m + 8n_Y + 13n_X + 6m \\ &= 18m + 8n_Y + 13n_X \\ &\leq 31n \end{aligned}$$

which is strictly smaller than $\frac{\beta}{2}$. Even if there exists some 2-gadget-respecting $\lambda_\perp$ such that every vertex in $G$ not belonging to a U2G, C2G, or U3G is necessarily included in every TaRDiS of $(G, \lambda_\perp)$, any such TaRDiS would still have cardinality less than $\#_{\text{U2G}} \cdot \frac{\alpha+4}{3} + \#_{\text{C2G}} \cdot \frac{\alpha+4}{3} + \#_{\text{U3G}} \cdot (\beta)$ and hence less than $k - 1$. Recall we assigned $k = \#_{\text{U2G}} \cdot \left(\frac{\alpha+1}{3}\right) + \#_{\text{C2G}} \cdot \left(\frac{\alpha+4}{3}\right) + \#_{\text{U3G}} \cdot (\beta) + 2m + 3n_X + 3n_Y + 1$, and that by our choice of $\beta$ we have $\frac{\beta}{2} > 2m + 3n_X + 3n_Y + 1$. □

**Lemma 3.39.** Any happy temporal assignment $\lambda$ such that every TaRDiS on $(G, \lambda)$ has size at least $k - 1$ is *nice*.

*Proof.* By Lemmas 3.36 and 3.38, any happy temporal assignment $\lambda$ such that every TaRDiS on $(G, \lambda)$ has size at least $k-1$ is 2-gadget and 3-gadget-respecting. Observe that if $\lambda$ is happy, 2- and 3-gadget-respecting, it must also be 1-gadget-respecting by the edge coloring constraint. Hence $\lambda$ is *nice*. □

We say that a nice function $\lambda$ *encodes* a truth assignment to the variables of $X$, and define this assignment as follows: let variable $x_i \in X$ be set to `True` if $\lambda(x_i^1, x_i^2) = 1$ and `False` otherwise. In Figure 3.13, `True` corresponds to the case where $A = 1$.

**Lemma 3.40.** A nice function $\lambda$ encodes an assignment to $X$ under which $\Phi(X,Y)$ is satisfiable if and only if $(G,\lambda)$ admits a TaRDiS of size $k-1$.

*Proof.* We first prove the forward direction. That is, if $\lambda$ encodes an assignment $\mathcal{X}$ to $X$ under which $\Phi(X,Y)$ is satisfiable then $(G,\lambda)$ admits a TaRDiS of size at most $k-1$.

In this case, there is an assignment $\mathcal{Y}$ to $Y$ such that $\Phi(\mathcal{X},\mathcal{Y})$ evaluates to `True`. We have, by Lemmas 3.34, 3.35 and 3.37, that exactly $\#_{\text{U2G}} \cdot \frac{\alpha+1}{3} + \#_{\text{C2G}} \cdot \frac{\alpha+4}{3} + \#_{\text{U3G}} \cdot \beta$ vertices are necessary and sufficient to cover all the vertices in U2G, U3G, C2G and U1G gadgets, and these vertices also reach every vertex adjacent to a C2G at time 2 exactly. Let $S$ include exactly those vertices.

For each $Y$-variable gadget we add the vertices $v_i^1$ and $T_i^1$ to $S$, if $y_i$ is `True` in $\mathcal{Y}$, and vertices $v_i^1$ and $F_i^1$ are added otherwise. Further, let $S$ include vertices $x_i^4$ and $x_i^{10}$ from each $X$-variable gadget. Note that irrespective of how that gadget is labeled under $\lambda$, this choice of vertices is sufficient to reach all vertices not already covered from some vertex in $S$ within a C2G. Additionally, note that now every literal vertex corresponding to a literal set to `True` under the combined assignment to $X \cup Y$ is reached from $S$. Since this is a satisfying assignment, every set $Q_j$ of clause gadget vertices as shown in Figure 3.12 must be incident to at least one literal vertex which is already reached from $S$. By symmetry we may assume this literal vertex is $u$ (otherwise, cycle the labels $q_i^1, \ldots, q_i^6$ such that it is). Then let $S$ include vertices $q_j^3$ and $q_j^5$. Observe that $S$ is a TaRDiS of size precisely $k-1$; all gadget vertices, literal vertices, clause vertices and variable vertices are reachable from $S$.

We now prove the other direction. Namely, if $(G,\lambda)$ admits a TaRDiS of size at most $k-1$ then $\lambda$ encodes an assignment to $X$ under which $\Phi(X,Y)$ is satisfiable. Recall that, by Lemma 3.39, $\lambda$ must be nice. Let $S$ be a minimum TaRDiS of size at most $k-1$ in $(G,\lambda)$. We may assume, by Lemma 3.4, that $S$ is canonical.

Since $S$ is minimum, we have, by Lemmas 3.34, 3.35 and 3.37, that exactly $\#_{\text{U2G}} \cdot \frac{\alpha+1}{3} + \#_{\text{C2G}} \cdot \frac{\alpha+4}{3} + \#_{\text{U3G}} \cdot \beta$ vertices in $S$ cover all the vertices in U2G, U3G, C2G and U1G gadgets. Then at most $2m + 3n_X + 3n_Y$ vertices of $S$ are not among these, and must be sufficient to cover the remaining vertices of $G$.

We define from $S$ a truth assignment to $Y$ as follows: if vertex $T_i^1$ or $T_i^2$ is in $S$, then we assign $y_i$ to be `True`, and we assign it to be `False` otherwise. We argue that this assignment together with $\mathcal{X}$ necessarily satisfies $\Phi(X,Y)$. Note that, since $S$ is canonical, it must contain at least 2 vertices from every clause gadget, 2 vertices from every $Y$-variable gadget, and 2 vertices from every $X$-variable gadget. All these bounds must be tight, else $S$ has cardinality more than $k-1$.

Suppose for contradiction that $\Phi(X,Y)$ is not satisfied by the assignment. Then some clause $c_j$ contains only False literals under the assignment, and at least 3 vertices in $Q_j$ must be in $S$, contradicting that at most 2 vertices from any clause gadget may be in the TaRDiS. Hence there is an assignment to $Y$ satisfying $\Phi(X, Y)$. □

**Lemma 3.41.** co-RPS is polynomial-time reducible to Happy MaxMinTaRDiS.

*Proof.* The construction above can be achieved in polynomial time. We have shown:

- The constructed MaxMinTaRDiS instance $(G, k, \tau = 3)$ is a yes-instance if and only if there is some *nice* $\lambda$ such that every TaRDiS for $(G, \lambda)$ has size at least $k - 1$ (Lemma 3.39).
- Every *nice* $\lambda$ encodes a truth assignment to $X$ under which $\Phi(X, Y)$ is satisfiable if and only if $(G, \lambda)$ admits a TaRDiS of size $k - 1$ (Lemma 3.40).

That is, $(G, k, \tau = 3)$ is a yes-instance of MaxMinTaRDiS if and only if $(X, Y, \Phi)$ is a yes-instance of co-RPS.

□

**Theorem 3.42.** Happy MaxMinTaRDiS is $\Sigma_2^P$-complete even restricted to inputs where $\tau = 3$ and the input graph $G$ is planar.

*Proof.* By Lemma 3.41 we have that co-RPS is polynomially reducible to Happy MaxMinTaRDiS. co-RPS is $\Sigma_2^P$-complete by Lemma 3.32. The reduction above preserves planarity of $\Phi$. We have containment of MaxMinTaRDiS in $\Sigma_2^P$ from Lemma 3.25. □

### 3.3.3 NP-completeness of Nonstrict MaxMinTaRDiS with lifetime 2

Here we consider the restriction of Nonstrict MaxMinTaRDiS to instances with lifetime 2. We show the problem to be equivalent to the Distance-3 Independent Set (D3IS) decision problem. We say that two problems $X$ and $Y$ are *equivalent* if they have the same language - that is, an instance $I$ is a yes-instance of $X$ if and only if the same instance $I$ is a yes-instance of $Y$. Where $X$ has a language consisting of triples $(G, k, \tau)$ and $Y$ has a language of tuples $(G, k)$, we may say that $Y$ is equivalent to $X$ with $\tau$ fixed to some value.

We define the distance $d(u, v)$ between two vertices $u, v$ in a temporal graph to be the number of edges in the shortest path between them in the footprint of the graph.

**Definition 3.43.** A distance-3 independent set (D3IS) of a static graph $H$ is a set $S \subseteq V(H)$ such that for all distinct $u, v \in S$, $d(u, v) \geq 3$.

The decision problem D3IS is defined as follows.

Distance-3 Independent Set (D3IS)
*Input:* A static graph $H = (V, E)$ and an integer $k$.
*Question:* Is there a set $S \subseteq V(H)$ of cardinality $k$ that is a distance-3 independent set?

We aim to show that a static graph $H$ and integer $k$ are a yes-instance of Nonstrict MaxMinTaRDiS with lifetime 2 if and only if the same graph $H$ and integer $k$ are a yes-instance of D3IS.

We begin by showing that existence of a maximal D3IS of size $k$ in a graph $H$ implies that we can find a temporal assignment $\lambda : E(H) \to \{1, 2\}$ such that a minimum TaRDiS in $(H, \lambda)$ is of cardinality $k$. Given such a D3IS $S$ of $H$, we assign $\lambda(u, v) = 1$, when $u \in S$ or $v \in S$, and $\lambda(u, v) = 2$, otherwise.

**Lemma 3.44.** Let $S$ be a maximal D3IS of a static graph $H$ and $\lambda$ be a temporal assignment of $H$ where $\lambda(u, v) = 1$ when $u \in S$ or $v \in S$, and $\lambda(u, v) = 2$ otherwise. Then $S$ is a minimum TaRDiS of $(H, \lambda)$.

*Proof.* We first show that $S$ is a TaRDiS. We assume without loss of generality that $H$ is a single connected component. Suppose for contradiction some vertex $u$ is not reachable from any vertex in $S$. Note that every vertex in $S$ trivially reaches its neighbours. So, by construction of $\lambda$, $u$ is incident only to edges at time 2. Since we have assumed that $H$ consists of a single connected component, there must be a static path in $H$ from each vertex in $S$ to $u$. Let $z$ be the closest vertex in $S$ to $u$. Then the shortest path from $z$ to $u$ must have length 2 and consist of an edge assigned time 1 followed by an edge assigned time 2. Else, $S$ is not maximal. Hence, the shortest path from $z$ to $u$ is a nonstrict temporal path and $u \in R_z$, contradicting our assumption. Thus $S$ is a TaRDiS of the constructed instance.

We now show minimality of $S$. By construction of $\lambda$, every vertex $v \in S$ is temporally reachable only from its closed neighbourhood $N[v]$. No temporal path originating outside $N[v]$ can include any edge incident to $v$ since any such path must contain an edge assigned time 2 before the final edge which is assigned time 1. Since $S$ forms a D3IS, $N[u] \cap N[v] = \emptyset$ for all $u$, $v \in S$. Therefore, for $(H, \lambda)$ to be temporal reachability dominated, there must at least be a vertex from the neighbourhood of each vertex in $S$. These are disjoint sets, so any TaRDiS must have cardinality at least $k$. Hence $S$ is a minimum TaRDiS of $(H, \lambda)$. □

**Definition 3.45** (Sole Reachability Set)**.** We define the *sole reachability set* of a vertex $v$ in a TaRDiS $S$ as the set $SR(\mathcal{G}, S, v) = R_v(\mathcal{G}) \setminus (\cup_{u \in S \setminus \{v\}} R_u(\mathcal{G}))$. Equivalently, it is the set of vertices reachable from $v$ and not any other vertex in $S$.

When $\mathcal{G}$ is clear from context, we write $SR(S, v)$ for $SR(S, \mathcal{G}, v)$. Note that, in a minimum TaRDiS, every vertex has a non-empty sole reachability set.

**Definition 3.46.** We call a TaRDiS $S$ on a temporal graph $\mathcal{G}$ *independent* if and only if every vertex in $S$ is in its own sole reachability set under $S$.

**Lemma 3.47.** If a temporal graph $\mathcal{G}$ admits an independent nonstrict TaRDiS $S$, then $S$ is a D3IS in the footprint graph $\mathcal{G}_\downarrow$.

*Proof.* Consider two vertices $u, v \in S$ at distance $d(u, v)$ from one another in $\mathcal{G}_\downarrow$. If $u$ and $v$ are adjacent, then $u$ and $v$ reach each other, which would contradict independence of $S$. If $d(u, v) = 2$ then there is at least one vertex $w \in N[u] \cap N[v]$, and either $\lambda(u, w) \leq \lambda(w, v)$ or $\lambda(u, w) > \lambda(w, v)$. So one of $u$ and $v$ must reach the other. Hence any two vertices in $S$ must be distance at least 3 from one another, the definition of a D3IS. □

**Lemma 3.48.** If a temporal graph $\mathcal{G}$ with lifetime 2 has a minimum nonstrict TaRDiS of cardinality $k$, then $\mathcal{G}_\downarrow$ admits a D3IS of size $k$.

*Proof.* Our proof is constructive; given a minimum TaRDiS $S$, we show existence of an *independent* minimum TaRDiS $S^*$ of equal cardinality and then apply Lemma 3.47.

First, we justify some simplifying assumptions about $\mathcal{G}$. Since TaRDiS can be computed independently in disconnected components of $\mathcal{G}_\downarrow$, we will assume $\mathcal{G}_\downarrow$ is connected. Further, if $E_1(\mathcal{G}) = \emptyset$ or $E_2(\mathcal{G}) = \emptyset$ we may choose any single vertex from $\mathcal{G}$ to be a minimum TaRDiS which is also independent; hence we assume $E_1(\mathcal{G}) \neq \emptyset$ and $E_2(\mathcal{G}) \neq \emptyset$. We also recall that $SR(S, x) \neq \emptyset$ for all $x \in S$ by minimality of $S$.

We construct $S^*$ by replacing every vertex $x$ in $S$ such that $x \notin SR(S, x)$ with some vertex $x^*$ with the property that $R_{x^*} = R_x$ and $x^* \in SR(S, x)$, as follows. Let $y \neq x$ be a vertex in $S$ such that $y$ reaches $x$. The path from $y$ to $x$ cannot arrive at time 1, else $S$ is not minimal as $R_x = R_y$ and $S \setminus \{x\}$ is a TaRDiS. Choose $x^*$ to be the closest vertex to $x$ in $SR(S, x)$. We know such a vertex exists by minimality of $S$. We claim that the path from $x$ to $x^*$ arrives at time 1 and so $R_x = R_{x^*}$. To see this, suppose otherwise. The path must begin with at least one edge at time 1, otherwise $x^*$ would be reachable from $y$. If the path arrives at time 2, then the last vertex on the path reached at time 1 is closer to $x$ than $x^*$, and is in $SR(S, x)$. Else, some other vertex in $S$ reaches $x^*$.

This concludes our construction of $S^*$ as an independent minimum TaRDiS. By Lemma 3.47, $S^*$ is also a D3IS. □

Combining Lemmas 3.44 and 3.48 gives us the following theorem.

**Theorem 3.49.** Nonstrict MaxMinTaRDiS with lifetime $\tau = 2$ is equivalent to Distance-3 Independent Set.

Interestingly, the same does not hold for $\tau \geq 3$. A counterexample is shown in Figure 3.4c, where the minimum TaRDiS is larger than the maximum D3IS of the footprint. The Petersen graph also gives us a 3-regular counterexample. We are able to leverage the known results [87, 105, 106] to obtain the following corollaries describing some (in)tractable cases of the problem when $\tau = 2$:

**Corollary 3.50.** Nonstrict MaxMinTaRDiS restricted to inputs with $\tau = 2$ remains NP-complete even when restricted to planar, bipartite graphs with maximum degree 3. Furthermore, the problem is $W[1]$-hard with respect to the parameter $k$, the size of a minimum TaRDiS, and is APX-hard on $r$-regular graphs for all integers $r \geq 3$.

**Corollary 3.51.** Nonstrict MaxMinTaRDiS restricted to inputs with $\tau = 2$ is in NP, admits a PTAS on planar graphs and is tractable on interval graphs, trapezoid graphs and circular arc graphs.

## 3.4 Parameterized complexity results for TaRDiS

In Section 3.2, we showed that the variants of TaRDiS are NP-complete and W[2]-hard with respect to $k$. This rules out $k$ as a candidate parameter for an fpt algorithm. We begin by showing inclusion of TaRDiS in FPT with respect to locally earliest edges and tractability when the input is heavily restricted. We then give an algorithm which solves each variant of TaRDiS on a nice tree decomposition of the footprint graph. This generalises the tree algorithm given in Section 3.2.

A problem $\Pi$ is said to be *fixed-parameter tractable* (fpt) with respect to some parameter $k$ if there is an algorithm solving $\Pi$ in time $f(k) \cdot \text{poly}(n)$ (where $n$ is the size of the instance of $\Pi$). The complexity class FPT consists of all problem-parameter pairs admitting an fpt algorithm.

### 3.4.1 FPT results with a restricted temporal assignment

We begin with two FPT results that require the temporal assignment to be restricted in some way to recover tractability of TaRDiS. In the first, we consider Happy TaRDiS and Nonstrict TaRDiS when parameterized by the number of locally earliest edges. Following that, we consider Strict TaRDiS when each component of the input is restricted in some way.

**Lemma 3.52.** Happy TaRDiS and Nonstrict TaRDiS are in FPT with respect to the number of locally earliest edges and weakly locally earliest edges, respectively.

*Proof.* It suffices to observe that any instance with $t$ (weakly) locally earliest edges trivially admits a TaRDiS of cardinality $t$ (Corollary 3.5). If $k \geq t$, we must have a yes-instance. Otherwise there are $\binom{t}{k}$ possibilities for a (weakly) canonical TaRDiS, each of which can be checked for validity in polynomial time, giving a runtime $\binom{t}{k} \cdot \text{poly}(n)$. □

**Lemma 3.53.** Strict TaRDiS is in FPT parameterized by maximum degree in $\mathcal{G}_\downarrow$, lifetime and $k$, and Happy TaRDiS is in FPT with parameters lifetime and $k$.

*Proof.* Any decidable language consisting of words of length at most some constant $c$ can be decided in constant time, hence is in $P$. Any strict temporal path in a temporal graph $\mathcal{G}$ has length at most $\tau$. This entails that, for all $v$, $|R_v^<| \leq 2\Delta^\tau$ where $\Delta$ is the maximum degree of $\mathcal{G}_\downarrow$. Hence any strict TaRDiS $S$ in $\mathcal{G}$ must satisfy $|S| \geq \frac{V(\mathcal{G})}{2\Delta^\tau}$. Therefore, no instance $(\mathcal{G}, k)$ satisfying $|V(\mathcal{G})| > 2k\Delta^\tau$ can be in the language Strict TaRDiS. Note that, in a happy temporal graph, we can can apply the property that $\tau \geq \Delta$. Therefore, both problems restricted as above have constantly many yes-instances, each of size bounded by a constant. □

## 3.4.2 Preliminaries: treewidth and tree decompositions

We refer the interested reader to Chapter 10 of Niedermeier's *Invitation to Fixed-Parameter Algorithms* [117] for a fuller introduction. Definitions and results in this subsection are taken or adapted from that work.

**Definition 3.54** (Tree Decomposition, Treewidth)**.** We say a pair $(T, B)$ is a *tree decomposition* of $G$ if $T$ is a tree and $B = \{B(s) : s \in V(T)\}$ is a collection of subsets of $V(G)$, called *bags*, satisfying:

1. $V(G) = \cup_{s \in V(T)} B(s)$.
2. $\forall (u, v) \in E(G) : \exists s \in V(T) : \{u, v\} \in B(s)$. That is, for each edge in the graph, there is at least one bag containing both of its endpoints.
3. $\forall v \in V(G) : T[\{s : v \in B(s)\}]$ is connected; for each vertex, the subgraph obtained by deleting every node not containing $v$ in its bag from $T$ is connected.

The *width* of a tree decomposition is defined to be $\max\{|B(s)| : s \in V(T)\} - 1$. The *treewidth* of a graph $G$ is the minimum $\omega$ such that $G$ has a tree decomposition of with $\omega$.

For a given tree decomposition $(T, B)$ of graph $G$, we denote by $V_s \subseteq V(G)$ the set of vertices in $G$ that occur in bags of the subtree of $T$ rooted at $s$. It is a well-known result by Bodlaender that finding a tree decomposition of width at most $\omega$, if one exists, is in FPT with parameter $\omega$.

**Theorem 3.55** (Bodlaender [118])**.** For all fixed constants $\omega \in \mathbb{N}$, there exists a linear time algorithm that tests whether a given graph $G = (V, E)$ has treewidth at most $\omega$, and if so outputs a tree decomposition of $G$ with treewidth at most $\omega$.

For ease, we describe our TaRDiS algorithm on a tree decomposition with additional structural properties.

**Definition 3.56** (Nice Tree Decomposition, Join/Introduce/Forget/Leaf Node)**.** A tree decomposition $(T, B)$ is called a *nice* tree decomposition if:

- every node of $T$ has at most two children;

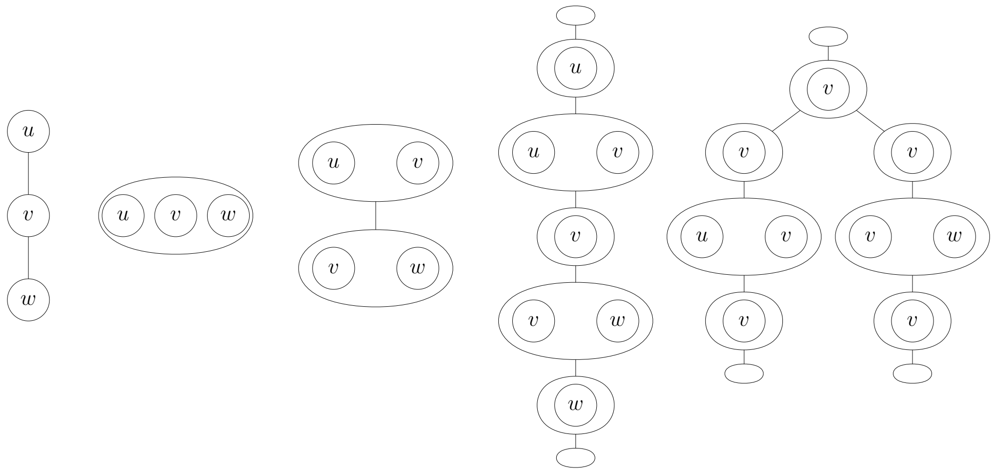


Figure 3.15: Graph $P_3$ and four of its tree decompositions. From left to right: $P_3$; the trivial decomposition of width 2; a decomposition of width 1; two possible nice tree decompositions of width 1. Note that every graph admits the trivial tree decomposition of width $V(G) - 1$, where there is only one bag $B(s) = V(G)$.

- if a node $s$ has two children $s_l$ and $s_r$, then $B(s) = B(s_l) = B(s_r)$, and we call $s$ a *join node*;
- if node $s$ has one child $s'$ then either:
  - $|B(s)| = |B(s')| + 1$ and $B(s) \supset B(s')$ ($s$ is an *introduce node*), or
  - $|B(s)| = |B(s')| - 1$ and $B(s) \subset B(s')$ ($s$ is a *forget node*);
- if node $s$ has no children then $B(s) = \emptyset$, and we call $t$ a *leaf node.*

An example of a graph and different tree decompositions of it can be found in Figure 3.15. It is a known result that a tree decomposition $(T, B)$ of a graph $G$ of width $\omega$ on $n$ nodes can be efficiently processed to obtain a nice tree decomposition $G$ of width $\omega$. We use this result as given below by Cygan, Fomin, Kowalik, Lokshtanov, Marx, Pilipczuk, Pilipczuk and Saurabh [88].

**Lemma 3.57** (Cygan [88])**.** If a static graph $H$ admits a tree decomposition of width at most $\omega$, then it also admits a nice tree decomposition of width at most $\omega$. Moreover, given a tree decomposition $(T, B)$ of width at most $\omega$, one can compute a nice tree decomposition of $H$ of width at most $\omega$ that has at most $O(\omega V(H))$ nodes in time $O(\omega^2 \max(V(T), V(H))$.

## 3.4.3 Algorithm for TaRDiS parameterized by treewidth and lifetime

The following gives an algorithm for TaRDiS given a nice tree decomposition $(T, B)$ of the footprint $\mathcal{G}_\downarrow$ of the input temporal graph $\mathcal{G}$. Specifically, it computes the cardinality of a minimum TaRDiS $S$. Note that we use the word "vertex" when referring to the original graph and "node" when discussing the decomposition graph. We use the symbols $\prec$ and $\succ$ as place-holders for strict/nonstrict inequalities. More specifically, in Strict TaRDiS we use $<$ and $>$ in the place of $\prec$ and $\succ$ respectively and for Nonstrict TaRDiS we use $\leq$ and $\geq$. By substituting the correct inequalities, it can be seen that the algorithm described is correct for each of Strict TaRDiS and Nonstrict TaRDiS.

Denote by $G_s$ the subgraph $(V_s, E_s)$, where $V_s$ (respectively $E_s$) is the set of all vertices (resp. edges) introduced in the subtree rooted at node $s$ of the decomposition tree. Intuitively, the algorithm works from leaf nodes to the root node and finds, at each node $s$, a partial solution consisting of a minimal TaRDiS $S$ for the subgraph $G_s$.

We define the *arrival time* of a temporal path at a vertex $v$ to be the time of the final time-edge of the path. For the trivial path from a vertex to itself, we say that the time of arrival is 0. Similarly, we define the *departure time* of a temporal path as the time of the first time-edge in the path. For example, the temporal path consisting of a single time-edge has the same departure and arrival time. We note that, for a path to be a strict temporal path, the arrival time at a vertex $v$ must be strictly before the departure time from $v$. We allow the arrival and departure times to be equal in nonstrict temporal paths of any length. We refer to a temporal path from a vertex $v$ to a vertex $u$ with earliest arrival time as a *foremost* temporal path. We call the arrival time of a foremost path the *foremost arrival time.* A foremost path from a set of vertices $S$ to a single vertex $v$ is a foremost path from a vertex $u$ in $S$ to $v$ such that the arrival time of a path from any other vertex in $S$ to $v$ is the same or later.

### States

A state of a bag $B(s)$ is a mapping $\psi : B(s) \to ([0,\tau] \cup \perp) \times ([0,\tau] \cup \perp)$ where $\psi(v) = (t_a(v), t_p(v))$, such that $t_a(v) \geq t_p(v)$ if both values are integers and $t_p(v) = \perp$ only if $t_a(v) = \perp$. Conceptually, we think of $t_a(v)$ as the arrival time of some path to $v$ from our partial TaRDiS and $t_p(v)$ as the time by which we "promise" arrival of such a path from the eventual full TaRDiS. The purpose of the promised arrival time is to ensure that all forgotten vertices are temporal reachability dominated by the TaRDiS. We use it by ensuring that, when a vertex is forgotten, there is a path arriving at the vertex by its promised arrival time from a second vertex in the bag which has a promised arrival time before the departure of the path. This means that if there exists a path from a vertex in the TaRDiS to the second vertex in the bag, we can append it with the path to ensure that the first is reached by the promised time. If no path between the vertices in the bag exists, the state can only be valid if the vertex is actually reached by its promised time from the partial TaRDiS (the arrival time is at most the promised time). Denote the set of all states of a node $s$ by $\Psi(s)$.

In our definition of a temporal graph, we assume that all edges are active at strictly positive times. Therefore, in both the strict and nonstrict variants of the problem, the earliest time any vertex can be temporally reachable from a vertex in the TaRDiS is 1, unless it is in the TaRDiS itself. Our intention is that, if $t_a(v) \neq \perp$ then $t_a(v)$ is exactly the time that $v$ is reached from some TaRDiS vertex. If $t_a(v) = 0$ for a vertex $v$ in the bag, this corresponds to $v$ being included in the partial TaRDiS $S$. We use the notation $t_a^{-1}$, $t_p^{-1}$ to denote the preimage of the functions $t_a$ and $t_p$ respectively.

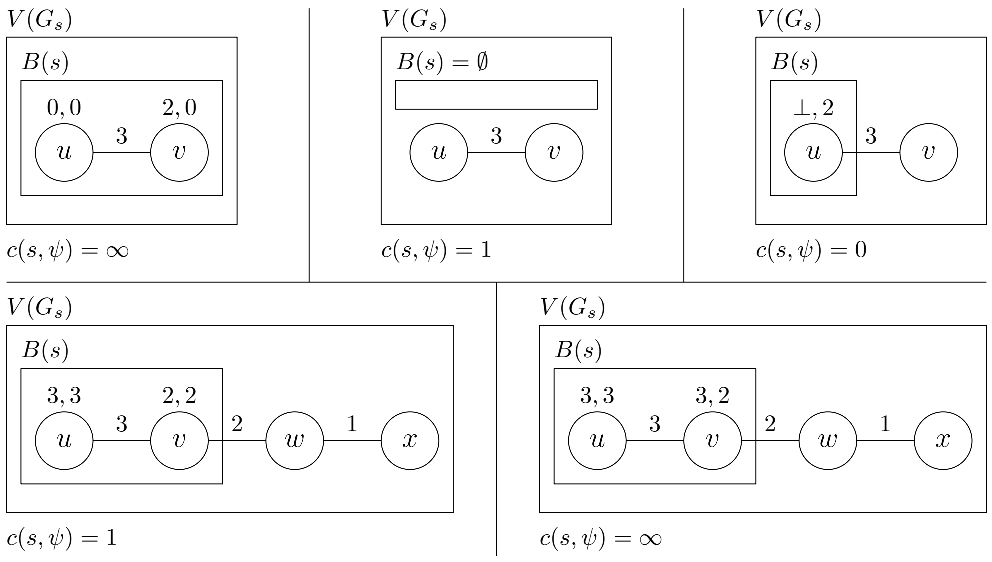


Figure 3.16: Five example bags, labelled with states and signatures. In the bottom right example, the infinite signature reflects the fact there exists no TaRDiS reaching $v$ at time 3 exactly.

That is, $t_a^{-1}(x)$ is the set of all vertices $v$ which are assigned $(x, t_p(v))$ under a state $\psi$.

We call a state $\psi$ of a bag $B(s)$ *consistent* if and only if there exists a set of vertices $S \subseteq V(G_s)$ such that

- for all $v \in B(s) \setminus t_a^{-1}(\perp)$ the foremost temporal path from some vertex in $S$ arrives at $t_a(v)$ exactly;
- for any vertex $w$ in $V(G_s) \setminus B(s)$ which is not reachable from some vertex in $S$, there is a temporal path to $w$ from some vertex $u$ in $B(s)$ with departure time $t$ such that $t \succ t_p(u)$.

We call such a set $S$ a set that *supports* $\psi$ of $s$. Note that, if $t_a(v) =\perp$, then it is possible to have a valid state where $v$ is reachable from some vertex in $S$.

## Signature

For a state $\psi$ of a node $s$, we define the *signature* $c(s,\psi)$ to be the cardinality of the smallest set $S^*$ which supports $\psi$ of $s$. If there is no such $S^*$, then we say that $c(s,\psi)=\infty$. We say that such a set $S^*$ *supports* the signature $c(s,\psi)$ if $S^*$ supports $\psi$ of $s$ and $|S^*| = c(s,\psi)$. The signature is a data structure with size $O(\tau^{2(\omega+1)})$ where $\omega$ is the width of the nice tree decomposition of $\mathcal{G}_\downarrow$. Note that two bags may contain the same (possibly empty) set of vertices. Therefore, the signature must be both a function of the state $\psi$ and of the node $s$ which $\psi$ is a state of. Examples of the signatures of different states can be seen in Figure 3.16.

We use the convention that the root node $r$ is empty, i.e. $B(r)=\emptyset$. Consequently there is only one possible state for the root node, namely the empty function, and hence $\Psi(r) = \{\emptyset\}$. Therefore, we have a yes-instance of TaRDiS if and only if $c(r,\emptyset) \leq k$. Note that $\emptyset$ is always a consistent state for $r$. It is supported by the trivial TaRDiS $S = V(G)$.

We now discuss how we iteratively calculate the signature of a state for each type of node in a nice tree decomposition.

### Leaf nodes

We assume that leaf nodes $l$ are empty, so there is only one trivial state, namely the empty function $\emptyset$. The signature of this state for leaves $l$ is $c(l, \emptyset) = 0$.

### Introduce Nodes

Let $s$ be an introduce node with child $s'$. Then we must have that $B(s) = B(s') \cup \{v\}$ for some $v \notin B(s')$. To describe how to find $c(s, \psi)$ for an introduce node $s$ and state $\psi$, we must define some new notation.

Let $\psi|_{B(s')}$ be the restriction of some state $\psi$ of $s$ to the bag $B(s')$. For a state $\psi$ and functions $g, f$, let the state $\psi^{t_a(A) \to g, t_p(B) \to h}$ be defined

$$\psi^{t_a(A) \to g, t_p(B) \to h}(x) := \begin{cases} (g(x), h(x)) & \text{if } x \in A \cap B \\ (g(x), t_p(x)) & \text{if } x \in A \setminus B \\ (t_a(x), h(x)) & \text{if } x \in B \setminus A \\ (t_a(x), t_p(x)), & \text{otherwise.} \end{cases}$$

For an introduce node $s$ with introduced vertex $v$ and state $\psi$ of $s$, we define $a_v$ to be the time of the earliest time-edge in $\{((v, u), t) \,|\, u \in B(s) \text{ and } t_a(u) \prec t \text{ under } \psi\}$. If no such edge exists, let $a_v = \infty$. Notionally, this is a time before which $v$ cannot be reached from a vertex in $S$ unless $v$ is itself in $S$. We define $R_v^t$ to be the set of vertices that are temporally reachable from $v$ by temporal paths which depart at a time $t' \succ t$. We use the convention that $R_w^\infty = R_w^\perp = \emptyset$ for all vertices $w$. For a vertex $u$ in $R_v^t$, define $\text{foremost}_v^t(u)$ to be the arrival time of a foremost path from $v$ to $u$ which departs at a time $t' \succ t$. An example of a valid state of an introduce node can be seen in Figure 3.17.

The following lemma finds the signature of a state of an introduce node using the signatures of the states of it's child based on 5 possible disjoint cases of the value of $t_a$ for the introduced vertex. In the first, a foremost path arrives at a vertex before it is required to by the state, so the state must be inconsistent. In the second case, the introduced vertex is added to the partial TaRDiS and the signature must be one more than a state of the child whose arrival times are not calculated using paths traversing the new vertex. The third and fourth cases occur when the introduced vertex could cause there to be a foremost path from the TaRDiS with an earlier arrival time. In the third, we deal with the situation where a pair of adjacent vertices both rely on each other to be reached from the TaRDiS by the time prescribed by the state. In this case, the signature must be the minimum of the signature of a state of the child such that each vertex in the pair is separately reached in time and the foremost arrival times do not traverse the introduced vertex. The fourth case occurs when the introduced vertex could change the arrival time of a foremost path, and we do not have the situation described in case 3. Here the signature of the state must be the same as the

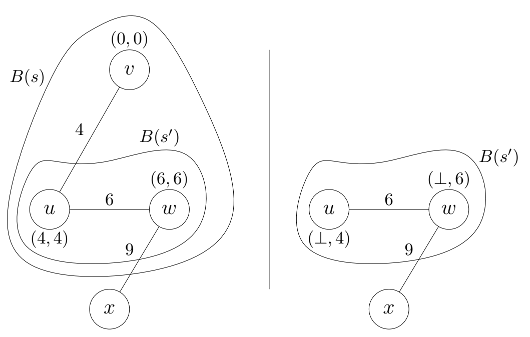


Figure 3.17: A valid state of an introduce node and a state of its child which are supported by the same partial TaRDiS. Here $x$ is a forgotten vertex.

signature of a state of the child such that the foremost arrival time at the vertices is found by only considering paths which do not traverse the introduced vertex. In the final case the state is inconsistent because $t_a$ is not equal to the foremost arrival time of a path from the TaRDiS to the introduced vertex.

**Lemma 3.58.** Let $\psi$ be a state of an introduce node $s$ with child $s'$ where $v$ is the vertex introduced. Define $Z$ to be the set $(R_v^{t_a(v)}(\mathcal{G}) \cap B(s)) \setminus \{u : \text{foremost}_v^{t_a(v)}(u) > t_a(u)\}$, and define the function $f(w) := \min\{t_p(w), \text{foremost}_v^{t_p(v)}(w)\}$. Then

1. if, for any $u \in R_v^{a_v}(\mathcal{G}) \cap B(s)$, $t_a(u) \neq \perp$ and $\text{foremost}_v^{a_v}(u) < t_a(u)$, then $c(s,\psi) = \infty$ (a foremost path arrives before it is prescribed to);
2. else, if $t_a(v) = 0$, $c(s,\psi) = 1 + c\left(s', \psi|_{B(s')}^{t_a(Z)\to\perp, t_p(B(s'))\to f(B(s'))}\right)$ (the introduced vertex is added to the partial TaRDiS);
3. else, if we allow nonstrict paths (i.e. if $1 \prec 1$), $a_v = t_a(v)$ and there exists a nonempty set $W$ of neighbours of $v$ where, for all $w \in W$, $t_a(w) = t_a(v) = \lambda(vw)$ then $c(s,\psi) = \min_{w\in W}\left\{c\left(s', \psi|_{B(s')}^{t_a(Z')\to\perp, t_p(B(s'))\to f(B(s'))}\right)\right\}$ for $Z' = Z \setminus \{w\}$ (a pair of neighbours each rely on the other to be reached on time);
4. else, if $t_a(v) \in \{a_v, \perp\}$ then $c(s,\psi) = c\left(s', \psi|_{B(s')}^{t_a(Z)\to\perp, t_p(B(s'))\to f(B(s'))}\right)$ (the introduced vertex could result in a foremost path with an earlier arrival time);
5. else, $c(s,\psi) = \infty$ (we have an inconsistent state).

*Proof.* We begin by noting that for any set $S$ supporting $\psi$ of $s$ if the foremost arrival time to $v$ from a vertex in $S$ is $t_a(v)$, then $\text{foremost}_v^{t_a(v)}(u) < t_a(u)$ contradicts that $t_a(u)$ gives the earliest time of arrival from a vertex in $S$ to $u$. Otherwise, we could append the temporal path to $v$ from $S$ with the path from $v$ to $u$ to find an temporal path with an earlier arrival time. Hence, in case (1) of the Lemma, $\psi$ is an inconsistent state.

We have characterised the value of $c(s,\psi)$ based on the value of $t_a(v)$ under $\psi$. This gives us four further cases ((2)-(5)) to consider, accounting for all possible values of $t_a(v)$. We show that our calculation of the signature $c(s,\psi)$ is correct for each possible value of $t_a(v)$ for the introduced vertex $v$. Let $S$ be a set that supports the signature of the state $\psi$ of node $s$. That is, it is a set of minimal cardinality such that $t_a^{-1}(0) = S \cap B(s)$; all vertices $w$ in $B(s) \setminus t_a^{-1}(\perp)$ are temporally reachable from $S$ by a foremost path arriving at $t_a(w)$; and each vertex in $G_s \setminus B(s)$ not reachable from $S$ is temporally reachable from a vertex $u$ in $B(s)$ departing at some time $t \succ t_p(u)$. Note that, for a consistent state, if there is a forgotten vertex reachable from a vertex $w$ in $B(s)$ by a path departing at time $t \succ t_p(w)$, the change in $t_p$ values ensures that this remains true in the child states over which we take the minimum value.

In case (2), $t_a(v) = 0$. Additionally, for all $u \in R_v^t$, $t_a(u) \leq \text{foremost}_v^{t_a(v)}(u)$ or $t_a(u) = \perp$, since otherwise we would have case (1). Hence for any set $S$ supporting $\psi$, $v \in S$ and all vertices in $R_v^0$ are reached by time $\text{foremost}_v^0(u)$. We claim that a set $S$ supports $\psi$ if and only if $S \setminus \{v\}$ supports the state $\psi' = (t'_a, t'_p) := \psi|_{B(s')}^{t_a(Z)\to\perp, t_p(B(s'))\to f(B(s'))}$ of $s'$. That is, every vertex $w$ in $B(s) \setminus R_v^0$ is temporally reached by a vertex in $S \setminus \{v\}$ at $t'_a(w)$ and every vertex in $V(G_s) \setminus B(s)$ that is not temporally reachable from $S \setminus \{v\}$ is temporally reachable from a vertex $y$ in $B(s')$ by a path departing at time $t' \succ t'_p(y)$. It is clear that, if states $\psi$ and $\psi'$ are valid, the former statement is true. This is because the value of $t_a$ for a vertex in $B(s) \setminus R_v^0$ is the same under both states.

To verify the latter statement, note that if a vertex $y'$ in $V(G_s) \setminus B(s)$ is reachable from a vertex $y$ in $B(s')$ by a path departing at $t' \succ t'_p(y)$ under $\psi'$, then the same must be true under $\psi$. By construction of a nice tree decomposition, there are no vertices in $V(G_s) \setminus B(s)$ that are adjacent to $v$. Therefore, any path from $v$ to a vertex in $V(G_s) \setminus B(s)$ must traverse another vertex $x$ in $B(s)$ and depart $x$ at time $t'' \succ t'_p(x)$ under $\psi'$. Thus, if a vertex $y'$ in $V(G_s) \setminus B(s)$ is reachable from a vertex $y$ in $B(s')$ by a path departing at $t' \succ t'_p(y)$ under $\psi$, then $y'$ must be reachable from a (possibly different) vertex $y''$ under $\psi'$. Therefore, a set $S$ supports $\psi$ if and only if $S \setminus \{v\}$ supports the state $\psi' = \psi|_{B(s')}^{t_a(Z)\to\perp, t_p(B(s'))\to f(B(s'))}$ of $s'$ and our calculation of $c(s, \psi)$ is correct.

Intuitively, cases (3) and (4) deal with the cases where the introduction of $v$ could change the earliest arrival time of a path from a vertex in $S$ to a vertex in $B(s) \setminus \{v\}$. In case (3) we deal with the possibility of nonstrict temporal paths when $t_a(w) = \text{foremost}_v^{t_a(v)}(w) = t_a(v)$ for some vertex $w$. In this case, a child state wherein $t_a(w) = \perp$ for every such $w$ may be supported by a set $S$ which does not support $\psi$. For this reason, we take the minimum over the signatures of states where the $t_a$ value of each such neighbour is unchanged. A set $S$ supports $\psi$ if and only if there exists a $w \in W$ such that $S$ supports $\psi|_{B(s')}^{t_a(Z')\to\perp, t_p(B(s'))\to f(B(s'))}$. The forward implication is clear from our description. Now suppose, for contradiction, that $S$ supports only $\psi|_{B(s')}^{t_a(Z')\to\perp, t_p(B(s'))\to f(B(s'))}$. Then there is either a vertex $u$ in $R_v^{t_a(v)}$ which is not reached from $S$ at time $t_a(u)$ or there is a forgotten vertex which is not reachable from $S$ or a vertex $w$ in $B(s)$ departing at $t' \succ t_p(w)$. Neither case is possible, and thus we have a contradiction.

In case (4), the above does not apply. This is either because we are in the strict setting or because there is no such nonempty set $W$, and $t_a(v) \in \{a_v, \perp\}$ under $\psi$. Then, for all vertices $w$ in $Z = (R_v^{a_v}(\mathcal{G}) \cap B(s)) \setminus \{u : \text{foremost}_v^{t_a(v)}(u) > t_a(u)\}$, there exists a foremost path from a set $S$ that supports the state which traverses $v$.

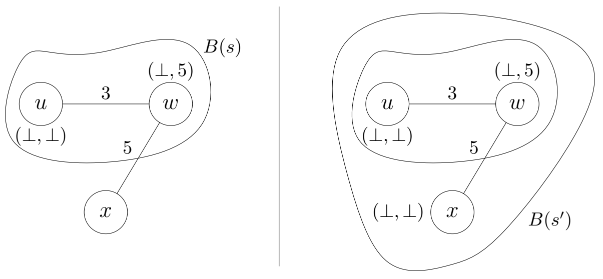


Figure 3.18: A forget node with a valid state and its extension to a state of its child supported by the same partial TaRDiS.

Therefore, any set $S$ which supports $\psi$ must support a child state where these vertices $w \in Z'$ have $t_a$ value $\perp$ and $t_p$ is updated as mentioned earlier. In addition, it is clear that any set which supports $\psi|_{B(s')}^{t_a(Z)\to\perp, t_p(B(s'))\to f(B(s'))}$ must support $\psi$. Since we have not added any vertices to $S$, the signature of $s$ under $\psi$ must be exactly the signature of $s'$ under $\psi|_{B(s')}^{t_a(Z)\to\perp, t_p(B(s'))\to f(B(s'))}$.

In case (5) we deal with all remaining possibilities, namely when $t_a(v)$ for the introduced vertex $v$ has a nonzero value and $t_a(v) \neq a_v$. Then $t_a(v)$ must be strictly before or after the earliest time-edge $((v, w), t)$ incident to $v$ such that, for the other endpoint $w$, $t_a(w) \prec t$. Therefore, for any set $S$ that supports $\psi$, if the foremost temporal path from a vertex in $S$ to each neighbour $w$ of $v$ arrives at $t_a(w)$, the foremost temporal path to $v$ from $S$ must arrive at a time which is not equal to $t_a(v)$. This implies that the state is inconsistent and must therefore have an infinite signature. □

**Forget Nodes**

Let $s$ be a forget node with child $s'$ such that $B(s) = B(s') \setminus \{v\}$. Let $\Psi_{\text{strong}} \subset \Psi(s')$ be the set of states which extend $\psi$ to $B(s')$ where $t_a(v) \in [0, \tau]$ and $t_a(v) = t_p(v)$. Intuitively, these are the states where our partial TaRDiS already reaches the forgotten node $v$ by the promised time $t_p(v)$, i.e. the promise is *strongly* satisfied.

Let $t'$ be the earliest time such that there exists a vertex $w$ in $B(s)$ and a temporal path in $B(s')$ from $w$ to $v$ departing at some time $t$ such that $t_p(w) \prec t$ under $\psi$ and arriving by $t'$. Let $\Psi_{\text{weak}} \subset \Psi(s')$ be the set of states which extend $\psi$ to $B(s')$ where either $t' \leq t_p(v)$ or $t_p(v) = \perp$. Intuitively, $\Psi_{\text{weak}}$ is the set of states where the requirement that a forgotten vertex is reached by a path departing at time $t \succ t_p(v)$ is automatically satisfied by a path from $w$ departing at $t \succ t_p(w)$ for some $w \neq v$. That is, the promise is *weakly* satisfied. An example of a valid state of a forget node can be seen in Figure 3.18.

**Lemma 3.59.** Let $s$ be a forget node with state $\psi$ and child $s'$ where $v$ is the vertex forgotten at $s$. Then

$$c(s, \psi) = \min\left(\{c(s', \psi') : \psi' \in \Psi_{\text{weak}} \cup \Psi_{\text{strong}}\} \cup \{\infty\}\right).$$

*Proof.* We begin by showing that if a set $S$ supports any state $\psi' \in \Psi_{\text{weak}} \cup \Psi_{\text{strong}}$ of $s'$ then the same set $S$ supports $\psi$. This shows us that the signature of $\psi$ is at most what we have calculated. Following this, we show that there is no smaller set that supports $\psi$ and hence that we have calculated the signature correctly.

Recall that set $S$ supports a state $\psi$ of a node $s$ if and only if, under $\psi$:

- For all vertices $u$ in $B(s) \setminus t_a^{-1}(\perp)$, the foremost path from $S$ arrives at time $t_a(u)$, and
- for all vertices $x$ in $G_s \setminus B(s)$ which are not reachable from $S$, there is a temporal path to $x$ which departs from a vertex $w$ in $B(s)$ at some time $t \succ t_p(w)$.

To start, consider the case where $S$ supports $\psi' \in \Psi_{\text{weak}}$. That is, there exists a vertex $w \in B(s')$ and a temporal path from $w$ to $v$ that departs at some time $t' \succ t_p(w)$ and arrives by time $t_p(v)$. As a result, any forgotten vertex $x$ temporally reachable from $v$ by a path departing at some time $t'' \succ t_p(v)$ must be temporally reachable from $w$ by a path departing at time $t'$ by prefixing the path from $v$ to $x$ with the temporal path from $w$ to $v$. If $\psi'$ is consistent, then all vertices $y$ in $B(s') \setminus \{v\} = B(s)$ that are reachable from a vertex in $S$ are reached by time $t_a(y)$ if $t_a(y) \neq \perp$. Therefore, the restriction of $\psi'$ to $B(s)$ is consistent and supported by $S$.

Now consider the case where $S$ supports a state $\psi' \in \Psi_{\text{strong}}$. Recall that $\Psi_{\text{strong}}$ is the set of states of $s'$ where $t_a(v) \in [0, \tau]$ and $t_a(v) = t_p(v)$. If such a state $\psi'$ is consistent for $s'$, then $v$ is reached from a vertex in $S$ by time $t_a(v)$. Using that $t_p(v) = t_a(v)$, we obtain that any forgotten vertex $x$ reachable from $B(s)$ by a path from $v$ departing at some time $t' \succ t_p(v)$ is reachable from $S$ as well. Namely, we can prefix the temporal path from $v$ to $x$ which departs at time $t'$ with the temporal path from $S$ to $v$ which arrives by time $t_a(v) = t_p(v)$. Hence we obtain that $\psi$ is consistent for $s$ and supported by $S$.

We now suppose for contradiction that our calculation of the signature is incorrect. We have already shown that the signature must be at most $|S|$ for some set $S$ supporting a state in $\Psi_{\text{strong}} \cup \Psi_{\text{weak}}$. Therefore, we suppose that there is a smaller set $S^*$ which supports $\psi$ whose intersection with $B(s)$ is exactly the set of vertices $t_a^{-1}(0)$ under $\psi$.

Consider the state $\psi^*$ extending $\psi$ to $B(s')$ in which $t_a(v)$ is the arrival time of the foremost path from $S^*$ to $v$ (or $\perp$ if there is no such path) and $t_p(v) = \tau$. Clearly $\psi^* \in \Psi_{\text{weak}} \cup \Psi_{\text{strong}}$, and $S^*$ supports $\psi^*$. This contradicts our assumption that $S^*$ was smaller than any set supporting a state in $\Psi_{\text{weak}} \cup \Psi_{\text{strong}}$. ☐

### Join Nodes

Let $s$ be a join node with children $s_1$ and $s_2$.

**Lemma 3.60.** Let $\psi_1 = (t_{a,1}, t_{p,1})$ and $\psi_2 = (t_{a,2}, t_{p,2})$ be states of the children $s_1$, $s_2$ of a join node $s$. We say that $\psi_1$ and $\psi_2$ *coincide* with the state $\psi$ if

- $\psi(v) = (0, 0)$ if and only if $\psi_1(v) = (0, 0)$ and $\psi_2(v) = (0, 0)$;

- for $i \in \{a, p\}$ and all $v \in B(s)$,

$$t_i(v) = \begin{cases} \min\{t_{i,1}(v), t_{i,2}(v)\} & \text{if } t_{i,1}(v), t_{i,2}(v) \neq \perp, \\ t_{i,1}(v) & \text{if } t_{i,2}(v) = \perp \neq t_{i,1}(v), \\ t_{i,2}(v) & \text{if } t_{i,1}(v) = \perp \neq t_{i,2}(v), \\ \perp, & \text{otherwise.} \end{cases}$$

Then we calculate $c(s, \psi)$ by

$$c(s, \psi) = \min_{\psi_1, \psi_2} \{c(s_1, \psi_1) + c(s_2, \psi_2) - |t_a^{-1}(0)|\}.$$

where the minimum is taken over all pairs of states $\psi_1$, $\psi_2$ which coincide with $\psi$.

*Proof.* We begin by showing that a state $\psi$ of a join node $s$ is consistent if and only if there are consistent states $\psi_1$ and $\psi_2$ of its children which coincide with $\psi$. We then give a proof of correctness of our calculation of the signature.

Suppose that the states $\psi_1$ and $\psi_2$ are consistent for the nodes $s_1$ and $s_2$ respectively and they coincide with the state $\psi$. Then, let $S$ be the union of sets $S_1$ and $S_2$ which support $\psi_1$ and $\psi_2$ respectively. By our assumption of consistency, all vertices in $V(G_s) \setminus B(s)$ are temporally reachable from a vertex in $S \cup B(s)$ and a foremost temporal path from a vertex in $S$ arrives at each vertex $u$ in $B(s)$ at time $t_a(u)$. In addition, if any forgotten vertex $w$ which is not reached from $S$ is temporally reachable from a vertex $v \in B(s)$ by a path departing at some time $t \succ t_{p,j}(v)$ for $j \in \{1, 2\}$, then $w$ must be temporally reached from $v$ by a temporal path departing at some time $t \succ t_p(v)$. Therefore, $\psi$ must be a consistent state of $s$.

We now assume that $\psi$ is a consistent state of $s$ supported by the set $S$. Suppose, for a contradiction, that $S$ supports $\psi$ and there are no child states supported by $S$ which coincide with $\psi$. There must be states for which $t_a$ describes the earliest time of arrival of a path from $S$ to any vertex in $B(s_1)$ and $B(s_2)$. Therefore, if there do not exist states $\psi_1$ and $\psi_2$ supported by $S$, then there must be a vertex $w$ in $G_{s_1} \setminus B(s_1)$ or $G_{s_2} \setminus B(s_2)$ which is not temporally reachable from a vertex in $S \cup B(s)$. If this is the case, then $\psi$ must also be inconsistent. This is because, by construction of a tree decomposition, the only (temporal) path from a vertex $G_{s_1} \setminus B(s_1)$ to $G_{s_2} \setminus B(s_2)$ or vice versa must traverse at least one vertex in $B(s) = B(s_1) = B(s_2)$. Therefore, if there is a temporal path from $S$ or $B(s)$ to each vertex in $G_s \setminus B(s)$, there must be a temporal path from $S \cup B(s_1)$ and $S \cap B(s_2)$ to each vertex in $G_{s_1} \setminus B(s_1)$ and $G_{s_2} \setminus B(s_2)$ respectively. Therefore, there exist child states supported by $S$.

The smallest set $S$ that supports a consistent state of $s$ must therefore be the smallest set that is the union of sets $S_1$ and $S_2$ which support consistent states $\psi_1$, $\psi_2$ respectively of its children $s_1$ and $s_2$ which coincide. To find the cardinality of this set, we employ the inclusion-exclusion principle. The only vertices which $G_{s_1}$ and $G_{s_2}$ have in common are those in $B(s)$. Therefore, $c(s, \psi) = \min_{\psi_1, \psi_2}\{c(s_1, \psi_1) + c(s_2, \psi_2) - |t_a^{-1}(0)|\}$ where the minimum is taken over pairs of states $\psi_1, \psi_2$ which coincide with $\psi$. $\square$

### Running Time and Extensions

We note that for every vertex $v$ in a bag, there are strictly fewer than $(\tau+2)^2$ values of $\psi(v)$. The number of vertices in a bag is bounded by treewidth $\omega$ plus 1. Thus, there are less than $(\tau+2)^{2(\omega+1)}$ states per node. We now explore the running time of calculating the signature at each type of node. For leaf nodes, the signature can only be one value; therefore this can be found in constant time. If we have the signatures of all child states of an introduce node and we want to verify validity of a given state of that node, we need to compute the value of $a_v$ for the introduced vertex $v$, the reachability set $R_v^{t_a(v)}$ and the earliest time of arrival at each other vertex in the bag from $v$ by a path departing at times $t \succ t_a(v)$ and $t' \succ t_p(v)$.

The reachability set $R_v^t(\mathcal{G})$ can be computed in time $O(n^2)$ by a modified breadth first search algorithm for any $v$ and $t$. Hence it takes $O(\tau n^3)$ time to calculate $R_v^t(\mathcal{G})$ for every vertex $v \in V$ and time $t \in [\tau]$. This can be done as a preprocessing step before we begin working up the tree decomposition.

The earliest time of arrival of the two paths can be computed using a variation of breadth-first search and thus takes $O(\omega^2)$ time. The value $a_v$ is computable in $O(\omega)$ time. In addition, for some cases in Nonstrict TaRDiS we find the minimum signature of restrictions of the state where we change the value of $t_a$ for some neighbours of $v$. Assuming all signatures of descendant nodes are calculated, this takes $O(\omega)$ time. Therefore, computing the signature of all states of an introduce node requires $O(\omega^2(\tau)^{2(\omega+1)})$ time.

For forget nodes, we must compare the signatures of multiple child states. There are $O(\tau^2)$ of these extensions for each state of the forget node. In addition, we compute the earliest time of arrival of a path from any vertex in $B(s)$ to $v$ in the subgraph induced by $B(s)$ with restrictions on its departure time. This can be achieved by a variant of a breadth-first search which takes at most $O(\omega^3)$ time. Thus, finding the signature of all states of a forget node requires $O(\omega^3(\tau)^{2\omega+4})$ time.

Finally, to compute the signature of the state of a join node, we must compare the states of both child nodes which coincide with this state. For a given vertex in the bag of a join node, the number of values of $\psi_1$ and $\psi_2$ which coincide with $\psi(v)$ are bounded by $O(\tau^4)$. Therefore, there are $O(\tau^{4(\omega+1)})$ tuples of states to consider when calculating the signature of $\psi$ on $s$. Therefore, we calculate the signature of a state of a join node in $O(\tau^{4(\omega+1)} \cdot \omega)$ time. Note that this dominates the time needed to generate all possible states for a given bag.

We now combine the lemmas from this section to get the following theorem. We can find a tree decomposition of width at most a constant $\omega$ if one exists in linear time by Theorem 3.55. Lemma 3.57 states that we can find a nice tree decomposition of constant width $\omega$ given a tree decomposition of width $\omega$ in linear time. We can recursively compute the signature of a given state of a node using Lemmas 3.58, 3.59 and 3.60. We solve TaRDiS by finding the signature $c(r, \psi)$ of the root where the state $\psi$ is the empty function which gives the cardinality of a minimal TaRDiS.

**Theorem 3.61.** The algorithm described takes as input a temporal graph $\mathcal{G}$ consisting of $n$ vertices with a nice tree decomposition of width at most $\omega$ and solves Strict and Nonstrict TaRDiS on $\mathcal{G}$ in time $O(\tau^{4(\omega+1)} \cdot \omega \cdot n + \tau n^3)$.

We emphasize that $\tau n^3$ is polynomial in the input size because $\tau \leq \mathcal{E}$ (recall, no snapshot in a temporal graph is empty). The algorithm allows for edges to be active multiple times. That is, it is not restricted to simple temporal graphs.

## 3.5 Parameterized complexity results for MaxMinTaRDiS

Having shown MaxMinTaRDiS is in $\Sigma_2^P$, finding instances of tractability is even more surprising than with the variants of the TaRDiS problem. We begin the following result, closely related to Lemma 3.53 which gives us tractability when each component of the input to Strict MaxMinTaRDiS is restricted.

**Lemma 3.62.** Strict MaxMinTaRDiS is in FPT when parameterized by maximum degree $\Delta$ in $H$ and $k$.

*Proof.* Recall from Lemma 3.30 that $(H, k)$ is a yes-instance of Strict MaxMinTaRDiS if and only if $(H, k-1)$ is a no-instance of Dominating Set. Also, any pair $(H, k)$ where $H$ has maximum degree $\Delta$ satisfying $|V(H)| > k(\Delta+1)$ is trivially a no-instance of Dominating Set. Applying the same reasoning as in our proof of Lemma 3.53, we obtain that Strict MaxMinTaRDiS is solvable in linear time when $\Delta$ and $k$ are bounded. If the input graph has at least $k(\Delta + 1)$ vertices then output YES, and otherwise solve the problem (necessarily of bounded size) by brute-force. $\square$

### 3.5.1 Containment in FPT with respect to treewidth and lifetime

We show tractability of our problems by expressing them in EMSO (extended monadic second order) logic and applying the variant of Courcelle's theorem given by Arnborg, Lagergren and Seese [119]. This result states that an optimisation problem which is definable in EMSO can be solved in polynomial time when parameterized by treewidth and length of the formula. The theorem is as follows.

**Theorem 3.63** (adapted from [120], Theorem 30)**.** Let $P$ be an EMSO-definable problem, then one can solve $P$ on graphs $G = (V, E)$ of order $n := |V|$ and treewidth at most $w$ in time $O(f_P(w) \cdot \text{poly}(n))$.

An EMSO formula over a static graph $H$ is a formula that uses:

1. the logical operators $\vee$, $\wedge$, $\neg$, $=$ and parentheses;
2. a finite set of variables, each of which takes an element of $V(H)$ or $E(H)$;
3. the quantifiers $\forall$ and $\exists$;
4. a finite set of variables which take subsets of the sets of edges or vertices;

5. integers.

We make use of the predicates $\neq$, $\implies$, $\impliedby$, $\iff$, $\subseteq$, $\in$ and $\setminus$, which can be implemented using the above. We note that any formula consisting only of the first 3 components is a first-order formula (FO). A formula which is first order with the addition of the fourth bullet point is referred to as an MSO formula. A more formal definition of an EMSO-definable problem is given by a survey by Langer, Reidl, Rossmanith and Sikdar [120].

**Theorem 3.64.** MaxMinTaRDiS is fixed-parameter tractable when parameterized by lifetime, $k$ and treewidth of the graph.

*Proof.* The formal definition of EMSO given by Langer Reidl, Rossmanith and Sikdar [120] requires a weight function bounded by a constant. Our weight function will be the cardinality of a TaRDiS, which is bounded by $k$. Since the temporal assignment is not part of the input, we encode it as a partition of edges into sets which correspond to the time at which they are active. The EMSO formula is constructed using the following auxiliary subformulae

- $\mathrm{card}(k, X)$ tests whether a set $X$ has cardinality at least $k$:
$$\mathrm{card}(k, X) := \exists x_1, \ldots, x_k \in X : \bigwedge_{1\leq i<j\leq k} x_i \neq x_j.$$
- $\mathrm{geq}(X_1, X_2)$ tests whether $|X_1| \geq |X_2|$ for sets $X_1$ and $X_2$:
$$\mathrm{geq}(X_1, X_2) := \exists k : \mathrm{card}(k, X_1) \wedge \neg\mathrm{card}(k + 1, X_2).$$
- $\mathrm{part}(S_1, \ldots, S_\tau)$ tests whether the sets of edges $S_1 \ldots S_\tau$ partition the edges of $H$:
$$\mathrm{part}(S_1, \ldots, S_\tau) := \forall e \in E :$$
$$\left(\bigvee_{1\leq i\leq\tau} e \in S_i \wedge \left(\bigwedge_{1\leq j<i} e \notin S_j\right) \wedge \left(\bigwedge_{i<\ell\leq\tau} e \notin S_\ell\right)\right).$$
The two right-most brackets can be ignored if we do not require that the temporal assignment is simple. To enforce a happy temporal assignment, we can add the requirement that no two edges in the same set share and endpoint. Recall that tractability of Happy MaxMinTaRDiS is shown with respect to lifetime and $k$ combined in Lemma 3.62. That is a stronger result than what we have here since Courcelle's theorem only gives tractability on graphs with bounded treewidth.

  The following subformulae can be adapted to write TaRDiS in MSO logic.
- $\mathrm{mconn}(X, S_t)$ tests whether the subgraph induced by $X$ in $G_t$ is connected; in particular, if it is true then all vertices in $X$ belong to the same connected component in $G_t$, and if $X$ is a connected component in $G_t$ then the predicate is true.
$$\mathrm{mconn}(X, S_t) := \forall Y \subset X, Y \neq \emptyset, \exists y \in Y, \exists x \in X \setminus Y : xy \in S_t$$

- mvconn$(v, w, t)$ tests whether two vertices are in the same connected component of $G_t$:

$$\text{mvconn}(v, w, t) := \exists X \subset V : v \in X \wedge w \in X \wedge \text{mconn}(X, S_t).$$

- mtadj tests whether two vertices $v$, $w \in V$ are adjacent at time $t$:

$$\text{mtadj}(v, w, S_t) := \exists e \in S_t : v \in e \wedge w \in e.$$

- mpath tests whether there is a temporal path from $v$ to $w \in V$ with latest time $\tau$:

$$\text{mpath}(v, w, S_1, \dots, S_\tau) := \exists v_0, \dots, v_\tau \in V :$$
$$v = v_0 \wedge v_\tau = w \wedge \bigwedge_{t=0}^{\tau-1} (v_t = v_{t+1} \vee \text{a}(v_t, v_{t+1}, S_{t+1}))$$

  where a$(v, w, S_t)$ can be substituted for mvconn$(v, w, t)$ or mtadj$(v, w, S_t)$ depending on whether we are testing for nonstrict or strict temporal paths respectively.

- TaRDiS$(S_1, \dots, S_\tau, X)$ which tests if every vertex is temporally reachable from $X$ by a temporal path of lifetime $\tau$:

$$\text{TaRDiS}(S_1, \dots, S_\tau, X) := \forall v \in V(H), \exists s \in X : \text{mpath}(s, v, S_1, \dots, S_\tau).$$

- $m$TaRDiS$(S_1, \dots, S_\tau, X)$ tests whether a set $X$ is a minimum TaRDiS:

$$m\text{TaRDiS}(S_1, \dots, S_\tau, X) := \forall X' \subset V(H) : \textsc{TaRDiS}(S_1, \dots, S_\tau, X) \wedge$$
$$(\textsc{TaRDiS}(S_1, \dots, S_\tau, X') \implies \text{geq}(X', X)).$$

- MinTaRDiS$(H, \tau, k)$ which tests if there is temporal assignment with lifetime at most $\tau$ such that there exists a minimum TaRDiS of size at least $k$ on $H$:

$$\textsc{MinTaRDiS}(H, \tau, k) := \exists X \subset V, \exists S_1, \dots, S_\tau \subset E :$$
$$\text{part}(S_1, \dots, S_\tau) \wedge m\textsc{TaRDiS}(S_1, \dots, S_\tau, X) \wedge \text{card}(k, X).$$

Therefore MaxMinTaRDiS can be expressed in EMSO and the theorem holds. □

Furthermore, we can use the expression to express TaRDiS in EMSO. This, however, gives a weaker tractability result than Theorem 3.61.

## 3.6 Conclusions and open questions

In this paper, we introduce the TaRDiS and MaxMinTaRDiS problems and study their (parameterized) complexity. We show a bound on the lifetime $\tau$ and a restriction to planar inputs combined are insufficient to obtain tractability (Theorems 3.12, 3.42, and Corollary 3.50) and moreover tightly characterize the minimum lifetime $\tau$ for which each problem becomes intractable. Further, we give an algorithm on a nice tree decomposition of a temporal graph which gives tractability of TaRDiS with respect to lifetime and treewidth of the footprint of the graph. In addition, we show that $\tau$, $k$ and the treewidth of the input graph combined are sufficient to yield tractability in all cases of MaxMinTaRDiS by leveraging Courcelle's theorem.

These results leave open the following questions:

**Question 3.1.** What is the exact complexity of Nonstrict MaxMinTaRDiS with lifetime $\tau \geq 3$?

**Question 3.2.** What is the exact complexity of Happy MaxMinTaRDiS with lifetime $\tau \geq 4$?

**Question 3.3.** Is there a structural parameter of the footprint graph (such as treewidth) which is alone sufficient to obtain tractability for any of the considered variants?

An interesting possible extension of our work would be to find approximability results for these problems. Another interesting dimension is the comparison of Nonstrict MaxMinTaRDiS and Happy MaxMinTaRDiS when $\tau$ is lower-bounded by a function of the number of edges $m$. With the constraint $\tau = m$ (and requiring that every $1 \leq t \leq \tau$ is used) Happy MaxMinTaRDiS becomes a subproblem of Nonstrict MaxMinTaRDiS, and their computational complexity in this case is an interesting open question. Analogously to $t$-Dominating Set [121], $t$-TaRDiS, in which $t$ individuals must be reached provides a natural generalization of our problem and the potential for parameterization by $t$. Recall that, in Strict MaxMinTaRDiS, it is always optimal to choose the constant function as our temporal assignment $\lambda$. It may be interesting to consider restrictions on $\lambda$ other than happiness which require the use of a non-constant temporal assignment (as in [81]) to make the problem more interesting.

# Chapter 4

# Reconfigurable Routing in Data Center Networks

## 4.1 Introduction

The rapid growth of cloud computing applications has induced demand for new technologies to optimize the performance of data center networks dealing with ever-larger workloads. The data center topology design problem (that of finding efficient data center topologies) has been studied extensively and resulted in myriad designs (see, e.g., [122]). Advances in hardware, such as optical switches reconfigurable in milli- to micro-seconds, have enabled the development of reconfigurable topologies (see, e.g., [123]). These topologies can adjust in response to demand (*demand-aware* reconfigurable topologies) or vary configurations over time according to a fixed protocol (*demand-oblivious* reconfigurable topologies; see, e.g., [124]). So-called *hybrid* data center networks are a combination of a static topology consisting of, for example, electrical switches, and a demand-aware reconfigurable topology implemented, for example, with optical circuit switches or free space optics (see, e.g., [33, 125, 34, 35]). An intuitive example of a simple reconfigurable topology is illustrated in Figure 4.1.

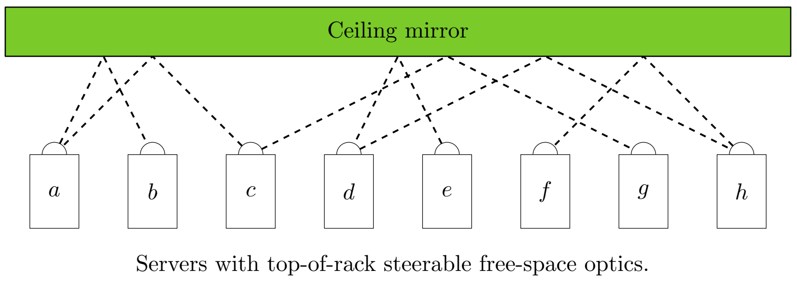


Figure 4.1: Basic model of an optical wireless data-center network, as described in [33, 34, 35]. Practical timescales for reconfiguration vary from milliseconds [34] to microseconds or nanoseconds [33, 35].

The hybrid network paradigm combines the robustness guarantees of static networks with the ability of demand-aware reconfigurable networks to serve large workloads at very low cost. Consider, for example, the hybrid network shown in Figure 4.2, and the configuration shown in Figure 4.3. In the (unaugmented) static network, there are two possible paths along which a message from node $b$ to node $d$ may be routed:

$b \to f \to h \to e \to d$ or $b \to f \to e \to d$. In the hybrid network as configured in Figure 4.3, the path $b \dashrightarrow a \to c \dashrightarrow d$ (among others) is an option[1].

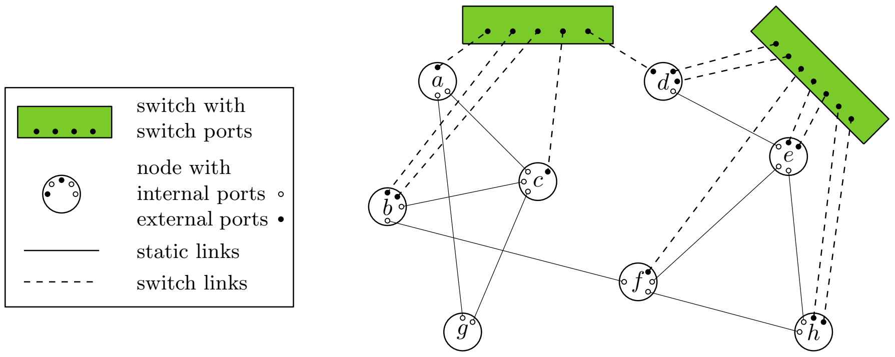


Figure 4.2: A hybrid network.

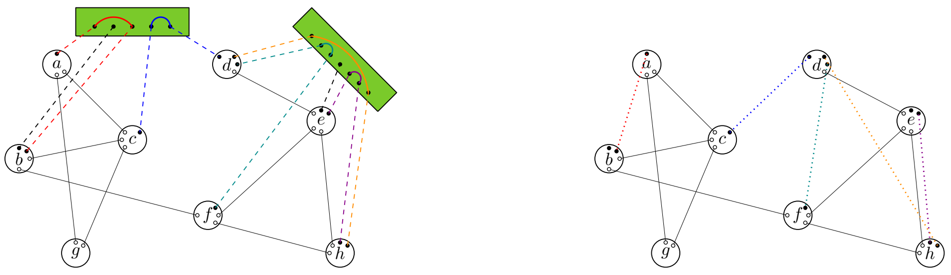


Figure 4.3: An augmented network and its abstracted dynamic links.

Of particular interest to us is the question of how the reconfigurable (optical) portion of the network should be configured for some demand pattern, formalized by Foerster, Ghobadi and Schmid [37] as the Reconfigurable Routing Problem (RRP): in short, given a hybrid network (consisting of a static network and of some switches) and a workload, we wish to choose a *configuration* (setting of the switches) which results in an optimal delivery of the workload.

Crucially, existing hardness results are only valid when the static network is allowed to be arbitrary, which is almost never the case in practice where interconnection and data center network design is driven by symmetry, high connectivity, recursive decomposition, and so forth. For example: the popular switch-centric data center network Fat-Tree [39] is derived from a folded Clos network; the server-centric data center network DCell [126] is recursively-structured whereby at each level, a graph-theoretic matching of servers is imposed; and the server-centric data center network BCube [40] is recursively-structured with a construction based around a generalized hypercube. (It should be noted that there do exist examples of unstructured data center networks, such as Jellyfish [127] and Xpander [128] which utilize the theory of random graphs.) Many (but not all) NP-complete problems become tractable when the input is restricted to the graphs providing the communications fabric for data center networks and other interconnection networks. For example, Hamiltonian paths are often trivial to find in many interconnection networks; indeed, no finite connected vertex-transitive graph *without* a Hamiltonian path is known to exist (the Lovász Conjecture contends there is no such graph - see Section 4 of [129]). This motivates

[1] We denote by $u \dashrightarrow v$ the concatenation of a switch link from $u$ to some switch, of the internal switch connection, and of a switch link to $v$ from that switch.

our investigation into how the complexity of RRP changes when we restrict to more structured and realistic networks. The question of the complexity of RRP for specific network topologies was specifically identified as an area for future work in [36].

In this chapter, we establish for the first time hardness results for RRP that apply to various specific families of highly structured static networks such as, for example, the hypercubes. Our constructions are (perhaps not surprisingly) of a much more involved nature than has hitherto been the case.

## 4.2 Problem Setting

The decision problem Reconfigurable Routing Problem considered in this chapter is a proper restriction of that presented in prior work [36, 37, 38]. In this section, we provide technical detail to fully formalize our version of the problem, but also additionally provide sufficient framing to briefly review existing results and to identify the areas strengthened by our contribution.

We adopt the usual terminology of graph theory though we tend to use 'nodes' and 'links' when speaking about the components of reconfigurable networks and 'nodes' and 'edges' when dealing with (abstract) graphs. We denote the natural numbers by $\mathbb{N}$ (we include $0 \in \mathbb{N}$) and the non-negative rationals by $\mathbb{Q}_+$.

### 4.2.1 Hybrid networks, (re)configurations and (segregated) routing

A hybrid network $G(S)$ can be visualized as in Figure 4.2, and consists of a static network $G$ and some switches $S$ augmenting it. A *static network* $G$ can be abstracted as an undirected graph $G = (V, E)$ so that each *static link* $(u, v) \in E$ has some fixed *weight* $w \in \mathbb{Q}_+$ (reflecting a transmission cost) and is incident with *internal ports* of two distinct nodes of $V$. The number of internal ports of some node $v \in V$ is then exactly the degree of $v$ in the abstracted graph $G$. We denote by $S$ a set of *switches* augmenting the static network $G$ with *switch links* joining *switch ports* of some switch to *external ports* of some of the nodes of $V$. Every switch link has weight 0 (we say more about switch link weights momentarily). Every switch $s \in S$ has at least two switch ports.

In general, the number of external ports of the nodes of a static network $G = (V, E)$ is variable, as is the number of switch ports of the switches of a hybrid network $G(S)$, and it may be the case that there is more than one switch link between a specific node and a specific switch. We assume that the switch links describe a bijection between the external ports and the switch ports; otherwise, there would be some unused ports, which we can safely ignore.

Given a hybrid network $G(S)$ and a switch $s \in S$, a *switch matching* $N_s$ of $s$ is a set of pairs of switch ports of $s$ so that all switch ports involved are distinct. Each switch matching represents an internal setting of the switch and naturally yields a set of pairs of external ports of nodes where all such ports are distinct; we refer to a set of pairs of external ports obtained in this way as a *node matching* (note that this differs from the standard graph-theoretic notion of a matching). An illustration of a configured hybrid network is shown in Figure 4.3: on the left side, switch matchings are represented as sets of arcs, and on the right side the corresponding node matching is shown as a set of dotted lines.

A *configuration* $N$ is a set of switch matchings, one for each switch. A configuration straightforwardly encodes the corresponding node matchings. We say that $(u, v)$ is a *dynamic link* in the configuration $N$ (we sometimes write $(u, v) \in N$) if $(u, v)$ appears in any node matching corresponding to $N$.

We allocate a fixed weight $\mu \in \mathbb{Q}_+$ to each internal port-to-port connection in a switch $s$. Although a dynamic link is an atomic entity, it can be visualized as consisting of a switch link followed by an internal port-to-port connection in $s$ followed by another switch link. We denote by $G(N)$ the static network $G$ augmented with the dynamic links (each of weight $\mu$) resulting from the configuration $N$ and we call $G(N)$ an *augmented network*. In the augmented network visualized in Figure 4.3, for example: $(a, b)$ is a dynamic link; $(a, c)$ is a static link; and $(e, h)$ is both a static link a dynamic link. Note that it is possible that an augmented network $G(N)$ is a multigraph.

The concepts defined above are driven by reconfigurable hardware technology such as optical switches, wireless (beamforming) and free-space optics, all of which establish port-to-port connections, i.e., switch matchings. The survey paper [125] provides some detail as regards the relationship between the emergent theoretical models and current opto-electronic technology.

### 4.2.2 Routing in hybrid networks

Consider again the example shown in Figure 4.3. In the configuration shown, a message $M$ from $c$ to node $e$ may be routed:

1. via static links only, along the path $\varphi_1 := c \to b \to f \to e$ with weight $3w$, or
2. via dynamic links only, along the path $\varphi_2 := c \dashrightarrow d \dashrightarrow h \dashrightarrow e$ with weight $3\mu$, or
3. via a combination of static and dynamic links, along the path $\varphi_3 := c \dashrightarrow d \to e$ with weight $\mu + w$.

Depending on the value of $\mu$, any of the paths may minimize the cost to route $M$: if $\mu \geq 2w$ then $\varphi_1$ is optimal; if $\mu \leq \frac{w}{2}$ then $\varphi_2$ is optimal; and if $\mu \in [\frac{w}{2}, 2w]$ then $\varphi_3$ is optimal. We may wish to bound the number of alternations allowed between optic and static links in any path a message takes; we capture this hardware requirement via a *segregation parameter* $\sigma \in \mathbb{N} \cup \{\infty\}$, as introduced in [38], that is the number of alternations between static and dynamic links. In the fully segregated case, $\sigma = 0$: messages may be routed either by static links only (as in $\varphi_1$) or by dynamic links only (as in $\varphi_2$). In the non-segregated case, $\sigma = \infty$ and there is no restriction on the number of alternations, so any path is admitted. Note $\varphi_3$ is admitted as a valid path to route $M$ if and only if $\sigma \geq 1$. The *dynamic link limit* $\delta$, like the segregation parameter $\sigma$, is a restriction on admissible flow-paths. Whereas $\sigma$ describes the maximum number of alternations between static and dynamic links permitted, $\delta$ describes the maximum *number* of dynamic links any flow-path may use. In particular, when $\delta = 1$ every flow-path must contain at most one dynamic link.

Networks are expected to route many messages (of varying sizes) optimally at the same time. Given a hybrid network $G(S)$ we represent the set of all demands we must optimize for as a *workload* (*matrix*) $D$ with entries $\{D[u,v] \in \mathbb{Q}_+ : u, v \in V\}$ providing the intended pairwise *node-to-node workloads* (each $D[u,u]$ is necessarily 0).

Given a configuration $N$ and $u, v \in V$ for which $D[u,v] > 0$, we route the corresponding workload via a path in $G(N)$ from $u$ to $v$ in $G(N)$ so that this chosen *flow-path* $\varphi(u,v)$ has *workload cost* $D[u,v] \times wt_{G(N)}(\varphi(u,v))$, where the *weight* $wt_{G(N)}(\varphi(u,v))$ is the sum of the weights of the links of the flow-path $\varphi(u,v)$ (if $G(N)$ has both a static link $(x,y)$ and a dynamic link $(x,y)$ then we need to say which we are using in $\varphi(u,v)$). The *total workload cost* (of $D$ under $N$) is defined as

$$\sum_{u,v \in V, D[u,v]>0} D[u,v] \times wt_{G(N)}(\varphi(u,v)).$$

Our aim will be to find a configuration $N$ in some hybrid network $G(S)$ and flow-paths in $G(N)$ for which the total workload cost of some workload matrix $D$ is minimized. In an unrestricted scenario, we would choose any flow-path $\varphi(u,v)$ to be a flow-path of minimum weight from $u$ to $v$ in $G(N)$, the weight of which we denote by $wt_{G(N)}(u,v)$. When $\sigma \neq \infty$ we must also ensure the flow-path has at most $\sigma$ alternations. We also have the analogous concepts $wt_G(\varphi(u,v))$ and $wt_G(u,v)$ where we work entirely in the static network $G$. Note that we often describe $D$ by a weighted digraph, which we usually call $D'$, so that the node set is $V$ and there is an edge $(u,v)$ of weight $w > 0$ if, and only if, $D[u,v] = w$. We also refer to some $D[u,v] > 0$ as a *demand* (from $u$ to $v$).

## 4.2.3 The Reconfigurable Routing Problem

We are now in a position to introduce our protagonist:

Reconfigurable Routing Problem ($\sigma$) (RRP($\sigma$))
*Input:* $(G, S, \mu, w, D, \kappa)$: $D$ is a workload matrix for the hybrid network $G(S)$ with static (resp. dynamic) links all of weight $w$ (resp. $\mu$).
*Question:* Does $G(S)$ admit some configuration $N$ such that the total workload cost of $D$ under $N$ (where the number of alternations for any path is bounded by $\sigma$) is at most $\kappa$?

As previously alluded to, this setting is more expressive than we require for most of this chapter, and more restrictive than the exact formalism considered in prior work [36, 37, 38]: in those works, $w$ and $\mu$ are sometimes allowed to be functions of their endpoints rather than fixed constants. This provides much more expressivity; notably, their model loses no power when it is restricted to inputs where $G$ is a complete graph and there is only one switch, since it is possible to simulate any other instance by assigning prohibitively large weights to any static edges and any pair of switch ports which should not be usable.

We now turn to the "realistic" networks we mentioned in our introduction. Henceforth unless otherwise specified, static link weights are all equal (and normalized to 1) and dynamic link weights are always some fixed constant $\mu \in \mathbb{Q}_+$. Also, there is a single switch and all nodes are connected to it with identical hardware. This is both practically relevant and intuitively realistic; see e.g. Figure 4.1. Then the set of switches $S$ of the hybrid network consists of just one switch, which is fully described by the number of switch links each node in the hybrid network has, which we call $\Delta_S$. This is closely related to the maximum reconfigurable degree $\Delta_R$ from [38], which is an upper bound on the number of external ports per node. The resulting restriction of RRP can be formalized as follows:

$\Delta_S$-SWITCHED RRP ($\sigma$)
*Input:* $(G, \mu, D, \kappa)$: $D$ is a workload matrix for the hybrid network $G(S)$ with static (resp. dynamic) links all have weight 1 (resp. $\mu$) (where $S$ consists of a single switch that every node in $G$ is connected to exactly $\Delta_S$ times).
*Question:* Does $G(S)$ admit some configuration $N$ such that the total workload cost of $D$ under $N$ (where the number of alternations for any path is bounded by $\sigma$) is at most $\kappa$?

## 4.3 Results

Table 4.1 shows a summary of hardness results from previous work as well as our three main intractability results. In general terms, we obtain NP-completeness for 2-SWITCHED RRP and 3-SWITCHED RRP on any fixed class of static networks of practical interest (defined more fully below) and for any value of $\sigma$. We then restrict our focus (and associated parameters) to the case where the static network is a hypercube when we establish the NP-completeness of 1-SWITCHED RRP($\sigma = 3$) in this setting; we conjecture that a similar construction can be used to establish hardness when $\sigma > 3$. We also, in Theorem 4.4, show that 1-SWITCHED RRP($\sigma = 0$) is solvable in polynomial time. The cases when $\sigma \in \{1, 2\}$ remain interesting open problems. Subsection 4.3.1 is devoted to the case of 1-SWITCHED RRP($\delta = 1$) (i.e. any flow-path must contain at most a single dynamic link) which entails at most 2 alternations, and we show this case is NP-complete for hypercubes, grids, and toroidal grids. The proof in that section is intended to facilitate the task of proving hardness for other interesting families of networks.

[2]By using variable $\mu$ with prohibitively large weights, it is possible to simulate many switches with just one.

| Result | $\lvert S\rvert$ | $\Delta_R$ | $\sigma/\delta$ | $D$ | link weights | notes |
|---|---|---|---|---|---|---|
| [36], Theorem 1 | $\Theta(n)$ (or 1 [2]) | $\Theta(n)$ | any $\sigma \geq 2$ | sparse, all values 0 or 1 | variable; $w \in [1, 100n^2]$ $\mu \in [1, 100n^2]$ | Showed inapprox. within $\Omega(\log n)$ |
| [37], Lemma 1 | $\Theta(n)$ | 1 | | | fixed; $w = \mu = 1$ | All switches have 3 ports. |
| [37], Theorem 2 | 1 | | | | | $G$ has $\Theta(n)$ components |
| [38], Theorems 4.1, 4.2 | | 2 | any $\sigma \geq 0$ | dense, values in poly$(n)$ | | $G$ is empty; there are no static links |
| Theorem 4.1 | | | | sparse, values in poly$(n)$ | fixed; $w = 1$ $\mu \in \Theta(\frac{1}{\text{poly}(n)})$ | $G \in \mathcal{H}$, where $\mathcal{H}$ is any polynomial family of networks (incl. hypercubes, grids, cycles). |
| Theorem 4.2 | | 3 | | | fixed; $w = 1$ $\mu \in \Theta(\frac{1}{\log(n)})$ | |
| Theorem 4.6 | | 1 | $\sigma = 3$ | | fixed; $w = 1$ any $\mu \in (0, 1)$ | $G$ is a hypercube |
| Theorem 4.19 | | | $\delta = 1$ | | | $G$ is a hypercube, grid, or toroidal grid |

Table 4.1: Settings for some pre-existing hardness results for RRP. $|S|$ is the number of switches; $\Delta_R$ is the maximum number of external ports per node; $\sigma$ is the segregation parameter and $\delta$ is the dynamic link limit; $D$ is the workload matrix; $n$ denotes the number of nodes in the instance.

As is standard in NP-hardness proofs, we reduce from known NP-complete problems to instances of RRP; the challenge is that, due to the expansive scope of our theorems, we lose several "degrees of freedom" which are used for encoding hard instances in, e.g., [130, 36, 37]. Specifically, we may not make use of varying static or dynamic link weights to prohibit certain connections, nor encode any features of the input instance in the topology of the hybrid network $G(S)$. For example, in Lemma 1 [37], many small switches with two feasible configurations each are used to encode a truth assignment, and in Theorem 1 of [36] "bad" links are given weights of order $\Theta(n^2)$. Neither of these mechanisms can be leveraged to obtain hardness in our setting; in this sense, our hardness results are strictly stronger and also harder to obtain than those from [130, 36, 37]. We are constrained to choose a *size* for the network $G$, and then to encode the input instance in the demand matrix $D$.

Our first two results hold for a wide class of graph families, which may be of broader interest for the study of computational hardness in network problems. Rather than allowing arbitrary static networks in instances of RRP, we wish to force any such static network to come from a fixed family of networks where a *family of networks* $\mathcal{H}$ is an infinite sequence of networks $\{H_i : i \geq 0\}$ so that the size $|H_i|$ of any $H_i$ is less than the size of $H_{i+1}$. However, we wish to control the sequence of network sizes. Consequently, we define a *polynomial family of networks* as being a family of networks $\mathcal{H} = \{H_i : i \geq 0\}$ where there exists a polynomial $p_{\mathcal{H}}(x)$ so that $|H_{i+1}| = p_{\mathcal{H}}(|H_i|)$, for each $i \geq 0$[3]. Note that given any $n \geq 0$, we can determine in time polynomial in $n$ the smallest $i$ such that $n \leq |H_i|$. As an example of a polynomial family of networks, consider the hypercubes; here, the polynomial $p_{\mathcal{H}}(x) = 2x$. Other examples include independent sets, complete graphs, cycles, complete binary trees and square grids,

[3]Technically, we insist that there exists a polynomial Turing machine $\mathcal{M}$ which computes $H_{i+1}$ on input $H_i$, for each $i \geq 0$, but this definition obfuscates the utility of this description.

among many others. The sweeping generality of having a single construction which holds for any polynomial family $\mathcal{H}$ poses a challenge in our proofs of Theorems 4.1 and 4.2; we require that our constructed network $H(S)$ behaves identically when $H$ is a connected (or even complete) graph and, at the opposite extreme, when $H$ is disconnected (or even independent).

**Theorem 4.1.** For any polynomial family of networks $\mathcal{H} = \{H_i : i \geq 0\}$, the problem 2-SWITCHED RRP restricted to instances $(H, \mu, D, \kappa)$ satisfying:

- $H \in \mathcal{H}$ has size $n$
- the workload matrix $D$ is sparse and all values in it are polynomial in $n$
- $\mu \in \Theta(\frac{1}{poly(n)})$ is fixed for all dynamic links

is NP-complete.

*Proof.* Note that 2-SWITCHED RRP as in the statement of the theorem is in NP as it can be straightforwardly reduced to an equivalent instance of the more general RRP, which is known to be in NP. We now build a polynomial-time reduction from the problem 3-MIN-BISECTION to our restricted version of RRP where the problem 3-MIN-BISECTION is defined as follows:

- instance of size $n$: a 3-regular graph $G = (V, E)$ on $n$ nodes and an integer $k \leq n^2$
- yes-instance: there exists a partition of $V$ into two disjoint subsets $A$ and $B$, each of size $\frac{n}{2}$, so that the set of edges incident with both a node in $A$ and a node in $B$ has size at most $k$; that is, $G$ has *bisection width* at most $k$.

Note that any 3-regular graph necessarily has an even number of nodes. The problem 3-MIN-BISECTION was proven to be NP-complete in [131] where it was also shown that approximating 3-MIN-BISECTION to within a constant factor approximation ratio entails the existence of a constant factor approximation algorithm for the more general and widely-studied problem MIN-BISECTION, which is defined as above but where $G$ can be arbitrary and where $A$ and $B$ have sizes differing by at most one. Given an arbitrary instance $(G = (V, E), k)$ of size $n$ of 3-MIN-BISECTION, we now build an instance $(H, \mu, D, \kappa)$ of 2-SWITCHED RRP. Moreover, we may assume that $k \leq \frac{n}{3} + 46$ as it was proven in [132] that every 3-regular graph has bisection width at most $\frac{n}{3} + 46$.

First, we define a weighted digraph $D' = (V', E')$ which will encode a description of our workload matrix $D$ via: there is a directed edge $(u, v)$ with weight $w$ if, and only if, there is a node-to-node workload of $w$ from $u$ to $v$. Let $\bar{n}$ be the size of the network $H_i$ where $i$ is the smallest integer such that $n + 6n^2 + 2 \leq |H_i|$ and set $H = H_i$.

- The node set $V'$ is taken as a disjoint copy of the node set $V$ of $G$, which we also refer to as $V$, together with the set of nodes $V_c = \{x_i, y_i : -\frac{L}{2} \leq i \leq \frac{L}{2}\}$, where $L = 3n^2$ (recall, $n$ is even), and another set of nodes $U$ of size $\bar{n} - (n + 6n^2 + 2)$; so, $|V'| = \bar{n}$. We call every node of $V_c$ a *chain-node*. For ease of presentation,

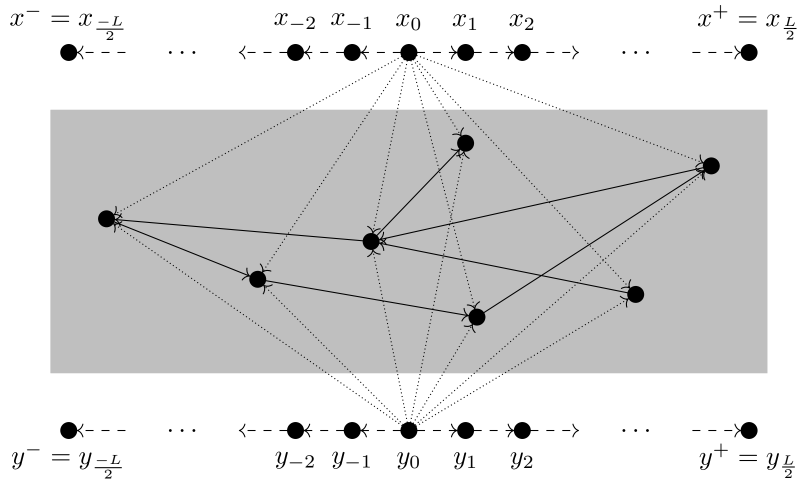


Figure 4.4: The graph $D'$.

we denote the chain-nodes $x_{\frac{L}{2}}$ and $x_{-\frac{L}{2}}$ by $x^+$ and $x^-$, respectively, and we define the chain-nodes $y^+$ and $y^-$ analogously.

- The (directed) edge set $E'$ consists of $E_\alpha \cup E_\beta \cup E_1$ where:
  - the set of *chain-edges* $E_\alpha = \{(x_i, x_{i+1}), (y_i, y_{i+1}) : 0 \leq i < \frac{L}{2}\} \cup \{(x_i, x_{i-1}), (y_i, y_{i-1}) : -\frac{L}{2} < i \leq 0\}$
  - the set of *star-edges* $E_\beta = \{(x_0, v) : v \in V\} \cup \{(y_0, v) : v \in V\}$
  - the set of *unit-edges* $E_1$ which is a copy of the edges $E$ of $G$, but on our (copied) node set $V$ and so that every edge is replaced by a directed edge of arbitrary orientation.

Note that the nodes of $U$ are all isolated in $D'$ and that $|V'| = \bar{n}$ (the nodes of $U$ will play no role in the following construction). The workloads on the edges of $E'$ are $\alpha$, $\beta$ or 1 depending upon whether the edge is a chain-edge from $E_\alpha$, a star-edge from $E_\beta$ or a unit-edge from $E_1$, respectively, where we define $\alpha = 24n^6$ and $\beta = 6n^3$. If the directed edge $(u, v)$ has weight $\alpha$ (resp. $\beta$, 1) in $D'$ then we say that $(u, v)$ is an $\alpha$-demand (resp. $\beta$-demand, 1-demand). The digraph $D'$ can be visualized as in Figure 4.4. The grey rectangle denotes the nodes of $V$, the nodes of $V_c$ appear along the top and the bottom and the dashed (resp. dotted, solid) directed edges depict the chain-edges (resp. star-edges, unit-edges). The nodes of $U$ are omitted.

As stated earlier, our static network $H$ is the network $H_i \in \mathcal{H}$ where $|H_i| = \bar{n}$. We refer to the node set of $H$ as $V'$ also and we refer to the subset of nodes within $V'$ corresponding to $V$ as $V$ also. Since we are in the 2-swtiched setting, we have one switch $s$ with $2|V'|$ ports so that every node of $H$ is adjacent, via switch links, to exactly two ports of the switch. Hence, our switch set is $S = \{s\}$ and our hybrid network is $H(S)$. It is important to note that for any configuration $N$, any node of $H(N)$ can be adjacent to at most 2 other nodes via dynamic links (as $\Delta_S = 2$).

As can be seen, we have the graph $G = (V, E)$, the digraph $D' = (V', E')$ and the hybrid network $H(S)$ with node set $V'$. Although $G$, $D'$ and $H(S)$ are disjoint in terms of node sets, we do not distinguish between, say, the node set $V$ of $G$ and the subset of nodes $V$ of $H$. It should always be obvious which set we are referring to.

We proceed similarly when we talk of specific nodes. As ever, we refer to 'edges' in graphs and 'links' in networks (but they are really one and the same).

We set the weight of any dynamic link as $\mu = \frac{1}{2L} = \frac{1}{6n^2}$ and the bound $\kappa$ for the total workload cost as $\kappa = \kappa_\alpha + \kappa_\beta + \kappa_1$ where:

- $\kappa_\alpha = 24n^6$
- $\kappa_\beta = 3n^4 + \frac{n^3}{2} + n^2$
- $\kappa_1 = \frac{k}{2} + \frac{1}{8} - \frac{1}{4n} + \frac{k}{3n^2}$.

The values of $\kappa_\alpha$, $\kappa_\beta$ and $\kappa_1$ have the following significance.

- Suppose that for every chain-edge $(x_i, x_{i+1})$ (resp. $(x_i, x_{i-1})$, $(y_i, y_{i+1})$, $(y_i, y_{i-1})$) of $E_\alpha$, we force a dynamic link joining $x_i$ and $x_{i+1}$ (resp. $x_i$ and $x_{i-1}$, $y_i$ and $y_{i+1}$, $y_i$ and $y_{i-1}$) in $H(N)$ and choose a corresponding flow-path serving this $\alpha$-demand as consisting of this dynamic link ($N$ is the resulting configuration from our chosen switch matching). The total workload cost of flow-paths serving $\alpha$-demands is $2L\alpha\mu = 24n^6 = \kappa_\alpha$.
- Further, suppose that the dynamic links incident with nodes of $V$ in $H(N)$ are chosen so that we have a path of dynamic links $p_A$ from $x^+$ to either $y^-$ or $y^+$, involving the subset of nodes $A \subseteq V$, and a path of dynamic links $p_B$ from $x^-$ to $y^+$ or $y^-$, respectively, involving the subset of nodes $B \subseteq V$, so that both $p_A$ and $p_B$ have length $\frac{n}{2} + 1$. That is, we choose the dynamic links so that they form a cycle $C$ (of length $n + 2L + 2$) in $H(N)$ covering exactly the nodes of $V$ and $V_c$. Suppose that for any star-edge $(x_0, v)$ (resp. $(y_0, v)$) of $E_\beta$, we choose the flow-path in $H(N)$ serving this star-edge as consisting entirely of dynamic links resulting from the shortest path in our cycle $C$ from $x_0$ to $v$ (resp. $y_0$ to $v$). The total workload cost of flow-paths corresponding to the star-edges is
$$4\mu\beta \sum_{i=1}^{\frac{n}{2}} (\frac{L}{2} + i) = 3n^4 + \frac{n^3}{2} + n^2 = \kappa_\beta.$$
- Further, suppose we choose the flow-path in $H(N)$ serving the 1-demand $(u, v)$ (in $E_1$) to be a path of dynamic links within the cycle $C$ of shortest length. If $u$ and $v$ both lie on $p_A$ or both lie on $p_B$ then the workload cost of this flow-path is at most $\mu(\frac{n}{2} - 1) = \frac{1}{6n}(\frac{1}{2} - \frac{1}{n})$, and if one of $u$ and $v$ lies on $p_A$ with the other node lying on $p_B$ then the workload cost of this flow-path is at most $\mu(\frac{n}{2} + L + 1) = \frac{1}{2} + \frac{1}{12n} + \frac{1}{6n^2}$. If the width of the bisection of $G$ formed by $A$ and $B$ is at most $k$ then the total workload cost of flow-paths corresponding to the unit-edges is at most
$$(\frac{3n}{2} - k)\mu(\frac{n}{2} - 1) + k\mu(\frac{n}{2} + L + 1) = \frac{k}{2} + \frac{1}{8} - \frac{1}{4n} + \frac{k}{3n^2} = \kappa_1.$$

From above, we immediately obtain that if $(G, k)$ is a yes-instance of 3-Min-Bisection then $(H(S), D, \mu, \kappa)$ is a yes-instance of RRP.

Conversely, suppose that $(H, \mu, D, \kappa)$ is a yes-instance of RRP. Let $N$ be a configuration (consisting of a switch matching of $s$) and let $F$ be a collection of flow-paths that witness that the total workload cost for the workload matrix $D$ is at most $\kappa$. Denote by $N$, also, the sub-network of $H(N)$ consisting solely of dynamic links. W.l.o.g. we may assume that every node of $H(N)$ is incident with exactly two dynamic links of $N$ (by adding additional dynamic links that we do not actually use in any flow-path, if necessary).

**Claim 4.1.1.** If $(u,v) \in E_\alpha$ (that is, $(u,v)$ is a chain-edge in $D'$) then $(u,v)$ is a dynamic link in $N$; so, the total workload cost of the flow-paths serving $\alpha$-demands is exactly $\kappa_\alpha = 24n^6$.

*Proof.* Suppose that $(x_i, x_{i+1}) \notin N$ (there is an analogous argument for each of $(x_i, x_{i-1})$, $(y_i, y_{i+1})$ and $(y_i, y_{i-1})$). So, the flow-path of $F$ serving the $\alpha$-demand $(x_i, x_{i+1})$ consists of at least two links and has workload cost at least $2\mu\alpha$ (as $2\mu$ is strictly less than 1 which is the weight of a static link). Hence, the total workload cost of flow-paths corresponding to the chain-edges is at least $\kappa_\alpha + \mu\alpha$. Denote the total workload cost of flow-paths serving the $\beta$-demands (resp. 1-demands) by $\bar{\kappa}_\beta$ (resp. $\bar{\kappa}_1$). So, $\kappa_\alpha + \mu\alpha + \bar{\kappa}_\beta + \bar{\kappa}_1 \leq \kappa$ with $\mu\alpha \leq \mu\alpha + \bar{\kappa}_\beta + \bar{\kappa}_1 \leq \kappa_\beta + \kappa_1$; that is, with $n^4 \leq \frac{n^3}{2} + n^2 + \frac{k}{2} + \frac{1}{8} - \frac{1}{4n} + \frac{k}{3n^2}$. This yields a contradiction (so long as $n$ is big enough) and the claim follows. □

**Claim 4.1.2.** The set of dynamic links $N$ forms a cycle in $H(N)$ covering exactly the nodes of $V \cup V_c$ and on which every link from $\{(x_i, x_{i+1}), (y_i, y_{i+1}) : 0 \leq i < \frac{L}{2}\} \cup \{(x_i, x_{i-1}), (y_i, y_{i-1}) : -\frac{L}{2} < i \leq 0\}$ lies. Moreover, every flow-path of $F$ serving a $\beta$-demand consists entirely of dynamic links.

*Proof.* By Claim 4.1.1, if $(u,v) \in E_\alpha$ then $(u,v) \in N$ and the total workload cost of the flow-paths serving the $\alpha$-demands is exactly $\kappa_\alpha = 24n^6$. Note that because every node of $H(N)$ is adjacent to at most 2 dynamic links, there are no other dynamic links incident with a node from $V_c \setminus \{x^+, x^-, y^+, y^-\}$.

Suppose that a flow-path of $F$ serving a $\beta$-demand consists entirely of dynamic links. So, the workload cost of such a flow-path is at least $\mu\beta(\frac{L}{2} + 1) = \frac{3n^3}{2} + n$. Alternatively, if a flow-path of $F$ serving a $\beta$-demand contains at least one static link then the workload cost of such a flow-path is at least $\beta = 6n^3$. Consequently, if at least one flow-path of $F$ serving a $\beta$-demand contains a static link (of weight 1) then the total workload cost of flow-paths serving the $\beta$-demands is at least $(2n-1)\mu\beta(\frac{L}{2}+1) + \beta = (2n-1)(\frac{3n^3}{2}+n) + 6n^3 = 3n^4 + \frac{9n^3}{2} + 2n^2 - n > 3n^4 + \frac{n^3}{2} + n^2 + \frac{k}{2} + \frac{1}{8} - \frac{1}{4n} + \frac{k}{3n^2} = \kappa_\beta + \kappa_1 = \kappa - \kappa_\alpha$ which yields a contradiction. Thus, all flow-paths of $F$ serving a $\beta$-demand consist entirely of dynamic links. In particular, $N$ must be connected. As $N$ is regular of degree 2 then $N$ must, in fact, be a cycle covering the nodes of $V \cup V_c$. The claim follows. □

By Claim 4.1.2, $N$ consists of a path of dynamic links involving all links from $\{(x_i, x_{i+1}) : -\frac{L}{2} \leq i < \frac{L}{2}\}$ from $x^-$ to $x^+$ concatenated with a path $p_A$ of dynamic links from $x^+$ to either $y^-$ or $y^+$ concatenated with a path of dynamic links involving all links from $\{(y_i, y_{i+1}) : -\frac{L}{2} \leq i < \frac{L}{2}\}$ from $y^-$ or $y^+$ to $y^+$ or $y^-$ concatenated with a path $p_B$ of dynamic links from $y^+$ or $y^-$ to $x^-$, respectively. W.l.o.g. we may assume that $p_A$ runs from $x^+$ to $y^+$ and $p_B$ from $y^-$ to $x^-$.

We can now recalculate the total workload cost of the flow-paths serving the $\beta$-demands. Suppose that there are $p$ nodes of $V$ on $p_A$ and so $n-p$ nodes of $V$ on $p_B$. We may assume that no nodes of $U$ lie on $p_A$ or $p_B$: if any do, then removing all such nodes from those paths results in a strictly smaller total workload cost. The total workload cost of the flow-paths serving the $\beta$-demands is

$$\begin{aligned}
&2\mu\beta\sum_{i=1}^{p}(\frac{L}{2}+i)+2\mu\beta\sum_{i=1}^{n-p}(\frac{L}{2}+i)\\
&= \quad p\mu\beta L+(n-p)\mu\beta L+2\mu\beta\sum_{i=1}^{p}i+2\mu\beta\sum_{i=1}^{n-p}i\\
&= \quad n\mu\beta L+(p(p+1)+(n-p)(n-p+1))\mu\beta\\
&= \quad 3n^4+n(2p^2-2np+n^2+n)\\
&\geq \quad 3n^4+\frac{n^3}{2}+n^2=\kappa_\beta \text{ (when } p \text{ takes the value } \tfrac{n}{2})
\end{aligned}$$

as the minimum value for $3n^4+n(2p^2-2np+n^2+n)$ is when $p=\frac{n}{2}$.

Suppose that the paths $p_A$ and $p_B$ have different lengths. From above, the total workload cost of the flow-paths serving the $\beta$-demands is at least $3n^4+n(2(\frac{n}{2}+1)^2-2n(\frac{n}{2}+1)+n^2+n) = 3n^4+\frac{n^3}{2}+n^2+2n = \kappa_\beta+2n$. Consequently, we must have that $2n \leq \kappa_1 = \frac{k}{2}+\frac{1}{8}-\frac{1}{4n}+\frac{k}{3n^2}$. As $k \leq \frac{n}{3}+46$ (from [132]), we have that $2n \leq \frac{n}{6}+23+\frac{1}{8}-\frac{1}{4n}+\frac{1}{9n}+\frac{46}{3n^2}$ which yields a contradiction. Hence, the paths $p_A$ and $p_B$ each have length $\frac{n}{2}$, with the total workload cost of the flow-paths corresponding to the unit-edges being at most $\kappa_1$.

Let $(u,v)$ be a unit-edge. From above: if $u$ and $v$ both lie on $p_A$ or both lie on $p_B$ and the flow-path serving the 1-demand $(u,v)$ consists entirely of dynamic links then the workload cost of this flow-path lies between $\mu$ and $\mu(\frac{n}{2}-1)$, i.e., $\frac{1}{6n^2}$ and $\frac{1}{6n}(\frac{1}{2}-\frac{1}{n})$; if $u$ and $v$ lie on $p_A$ and $p_B$, respectively, or vice versa, and the flow-path serving the 1-demand $(u,v)$ consists entirely of dynamic links then the workload cost of this flow-path lies between $\mu(L+2)$ and $\mu(\frac{n}{2}+L+1)$, i.e., $\frac{1}{2}+\frac{1}{3n^2}$ and $\frac{1}{2}+\frac{1}{12n}+\frac{1}{6n^2}$; and if the flow-path serving the 1-demand $(u,v)$ contains a static link then the workload cost of this flow-path is at least 1. In particular, we may assume that any flow-path corresponding to a unit-edge consists entirely of dynamic links.

Let $A$ (resp. $B$) be the nodes of $V$ that appear on $p_A$ (resp. $p_B$). So, $(A,B)$ is a bisection of $G$. Suppose that this bisection has width $k+\epsilon$, for some $\epsilon \geq 1$. So there are $k+\epsilon$ unit-edges whose corresponding work-flows contribute collectively at least $(k+\epsilon)\mu(L+2) \geq (k+1)(\frac{1}{2}+\frac{1}{3n^2}) = \frac{k}{2}+\frac{1}{2}+\frac{1}{3n^2}+\frac{k}{3n^2} > \frac{k}{2}+\frac{1}{8}-\frac{1}{4n}+\frac{k}{3n^2} = \kappa_1$ to the total workload cost, which yields a contradiction. Consequently, $G$ has bisection width at most $k$ and we have that if $(H,\mu,D,\kappa)$ is a yes-instance of RRP then $(G,k)$ is a yes-instance of 3-Min-Bisection. Our result follows as $(H,\mu,D,\kappa)$ can be constructed from $(G,k)$ in time polynomial in $n$. □

This result significantly strengthens Theorems 4.1 and 4.2 from [38]: there, RRP($\Delta_R \geq 2, \sigma = 0$) is shown to be NP-complete when the static network is an independent set, and the proof does not enable us to restrict the workload matrix $D$ meaningfully. The main weakness of Theorem 4.1 is its reliance on $\mu$ being a polynomial factor smaller than any static link weight. This is actually related to the fact that a connected 2-regular network, as is $G(N)$ when $G$ is an independent set and $\Delta_S = 2$, has diameter linear in the number of nodes $n$. A network of maximum degree 3, on the other hand, may have diameter logarithmic in $n$ (e.g., a complete binary tree has this property) and we indeed show NP-completeness of RRP($\Delta_S = 3$) when $\mu = \Theta(\frac{1}{\log n})$.

**Theorem 4.2.** For any polynomial family of networks $\mathcal{H} = \{H_i : i \geq 0\}$, the problem 3-SWITCHED RRP restricted to instances $(H, \mu, D, \kappa)$ satisfying:

- $H \in \mathcal{H}$ has size $n$
- the workload matrix $D$ is sparse and all values in it are polynomial in $n$
- $\mu \in \Theta(\frac{1}{\log(n)})$

is NP-complete.

*Proof.* As in the case of Theorem 4.1, membership of NP is straightforward. The problem RESTRICTED EXACT COVER BY 3-SETS (RXC3) is defined as follows:

- instance of size $n$: a finite set of *elements* $\mathcal{X} = \{x_i : 1 \leq i \leq 3n\}$, for some $n \geq 1$, and a collection $\mathcal{C}$ of 3-element subsets of $\mathcal{X}$, called *clauses*, where $\mathcal{C} = \{c_j : 1 \leq j \leq 3n\}$ and where each $c_j = \{x_1^j, x_2^j, x_3^j\}$ so that every element of $\mathcal{X}$ appears in exactly three clauses of $\mathcal{C}$
- yes-instance: $\mathcal{C}$ contains an *exact cover* $\mathcal{C}'$ for $\mathcal{X}$; that is, a subset $\mathcal{C}' \subseteq \mathcal{C}$ so that every element of $\mathcal{X}$ appears in exactly one clause of $\mathcal{C}'$ (hence, $\mathcal{C}'$ necessarily has size $n$).

The problem RXC3 is NP-complete as was proven in Theorem A.1 of [133]. Let $(\mathcal{X}, \mathcal{C})$ be an instance of RXC3 of size $n$. We will build an instance $(H, \mu, D, \kappa)$ of 3-SWITCHED RRP corresponding to $(\mathcal{X}, \mathcal{C})$.

First, we define a weighted graph $D'$ so as to describe the workload $D$. Having built $D'$, we will orient every undirected edge so that: there is a directed edge $(u, v)$ with weight $w$ if, and only if, there is a node-to-node workload of $w$ from $u$ to $v$ (we still refer to the resulting digraph as $D'$). Consequently, we will need one flow-path in our resulting hybrid network corresponding to each directed edge of $D' = (V', E')$. Our construction proceeds in stages.

- We begin with a complete binary tree $T$ of depth $d$ so that the number of leaves is $2^d$ with $2^{d-1} < n \leq 2^d$. If the leaves are $\{l_i : 1 \leq i \leq 2^d\}$ then we colour: every leaf $l_i$, where $n < i$, white; every leaf $l_i$, where $1 \leq i \leq n$, grey; and the root of $T$ grey. Throughout, if we do not specify the colour of a node as grey or white then it is coloured black. Note that $d = \lceil \log_2(n) \rceil$.

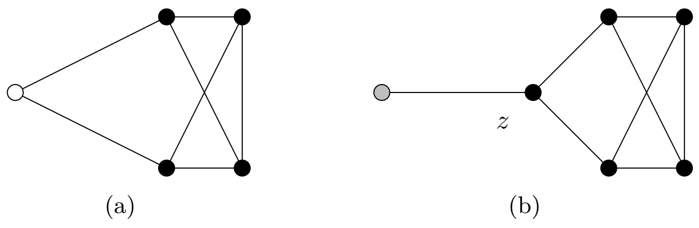


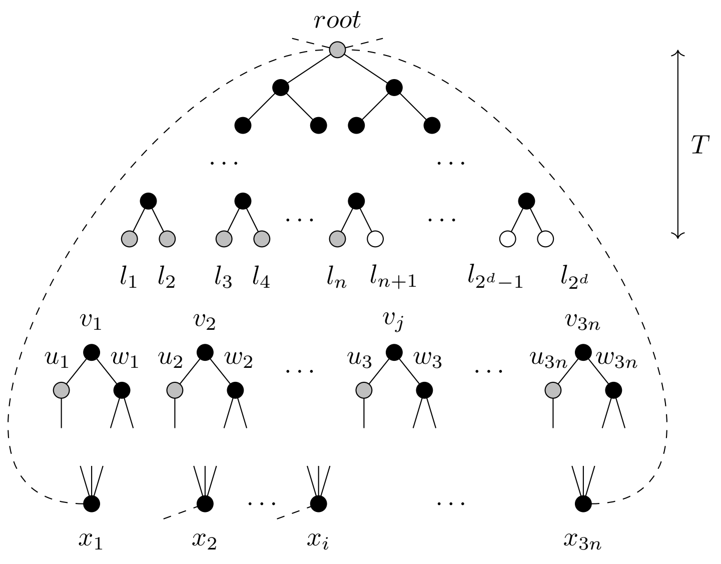
Figure 4.5: Attaching the gadgets.

- For every $x_i \in \mathcal{X}$, we create a unique node $x_i$ and for every clause $c_j \in \mathcal{C}$, we create a unique 2-path as follows: there are 3 nodes $u_j$, $v_j$ and $w_j$ together with the edges $(u_j, v_j)$ and $(v_j, w_j)$ and the node $u_j$ is coloured grey.
- For each $1 \leq j \leq 3n$, if the clause $c_j = \{x_1^j, x_2^j, x_3^j\}$ then there are edges $(x_1^j, u_j)$, $(x_2^j, w_j)$ and $(x_3^j, w_j)$.
- To every node coloured grey, we attach a unique gadget consisting of a 4-clique with one of its edges subdivided by a node, denoted $z$, so that there is an edge joining the gray node and the node $z$; and to every node coloured white, we attach the same gadget except that we identify the white node and the node denoted $z$. These attachments can be visualized as in Figure 4.5a and Figure 4.5b where the white and gray nodes are as shown.
- There is an edge $(r, x_i)$, for all $1 \leq i \leq 3n$: call these the *root-edges*.

This completes the construction of the graph $D'$ which can be visualized as in Figure 4.6 where the root-edges are depicted as dashed. Note that the number of nodes in $V'$ is $N = 6(2^d) + 28n + 4$ and the number of edges in $E'$ is $9(2^d) + 43n + 6$, with $3n$ of these edges root-edges. Let $D''$ be $D'$ with the root-edges removed. Note that every node of $D''$ has degree 3 except for the leaf nodes of $\{l_i : 1 \leq i \leq n\}$ and the nodes of $\{v_j : 1 \leq j \leq m\}$, all of which have degree 2 (we return to this comment presently). Let $E' = E'' \cup E^{root}$ where $E^{root}$ is the set of root-edges (and so $E''$ is the edge set of $D''$). As regards the edge-weights, orient each of the edges of $E'$ 'down' the graph $D'$ as it is portrayed in Figure 4.6 so as to obtain a digraph and: give each directed edge of $E''$ weight $\alpha$, where $\alpha = 4n\log_2(n)$; and give each directed root-edge of the form $(r, x_i)$ weight 1.

Let $\bar{n}$ be the size of the network $H_i$ where $i$ is the smallest integer such that $N \leq |H_i|$ and we fix our static network as $H = H_i$. We name a subset of nodes of $H$ as $V'$ with the remaining nodes named $U$. $\Delta_S = 3$, so we have one switch $s$ that has $3n$ switch ports and every node of $H$ has 3 external ports; so, every node of $H$ has exactly 3 switch links to $s$, and $S = \{s\}$. Finally, we define $\mu = \frac{1}{2\lceil\log_2(n)\rceil}$ and $\kappa = \mu\alpha|E''| + 3\mu n(\lceil\log_2(n)\rceil + 3)$ so as to complete the construction of our instance $(H, \mu, D, \kappa)$ of 3-SWITCHED RRP.

Suppose that $(\mathcal{X}, \mathcal{C})$ is a yes-instance of RXC3. So, let $\mathcal{C}' \subseteq \mathcal{C}$ be an exact cover of $\mathcal{X}$. For every $(u, v) \in E''$, we choose $(u, v)$ to be a dynamic link and ensure that every node of $\{v_j : 1 \leq j \leq 3n, c_j \in \mathcal{C}'\}$ is joined via a dynamic link to a leaf node of $\{l_i : 1 \leq i \leq n\}$. If $v_j$ is such that $c_j \in \mathcal{C}'$ then denote the unique leaf node to which there is a dynamic link from $v_j$ by $\bar{l}_j$. Denote the resulting switch matching



root
T
$l_1$ $l_2$ $l_3$ $l_4$ $l_n$ $l_{n+1}$ $l_{2^d-1}$ $l_{2^d}$
$v_1$ $v_2$ $v_j$ $v_{3n}$
$u_1$ $w_1$ $u_2$ $w_2$ $u_3$ $w_3$ $u_{3n}$ $w_{3n}$
$x_1$ $x_2$ $x_i$ $x_{3n}$

Figure 4.6: The graph $D'$.

of $s$ as constituting the configuration $N$. Referring back to our comment on node degrees above, the dynamic links so added must form a matching in $H(N)$. For any workload of weight $\alpha$ in $D$, we choose the corresponding flow-path to consist of the dynamic link joining the two nodes in question. Consequently, for all of the workloads corresponding to directed edges of $E''$, the total workload cost of the corresponding flow-paths is $\mu\alpha|E''|$. All of the remaining workloads involve the root and the nodes of $\{x_i : 1 \leq i \leq n\}$. Any one of these nodes $x_i$ is such that the element $x_i$ in a unique clause $c_j \in \mathcal{C}'$. We choose the flow-path consisting entirely of dynamic links that routes from the root down the tree $T$ to the leaf-node $\bar{l}_j$ and on to $v_j$ and, through either $u_j$ or $w_j$, on to $x_i$. This flow-path has (graph-theoretic) length $d+3$ and consequently the total workload cost due to flow-paths corresponding to edges of $E^{root}$ is exactly $3\mu n(d+3) = 3\mu n(\lceil \log_2(n) \rceil + 3)$. Hence, the total workload cost due to all flow-paths is $\mu\alpha|E''| + 3\mu n(\lceil \log_2(n) \rceil + 3) = \kappa$ and $(H, \mu, D, \kappa)$ is a yes-instance of 3-SWITCHED RRP.

Conversely, suppose that $(H, \mu, D, \kappa)$ is a yes-instance of 3-SWITCHED RRP and that $N$ is a configuration and $F$ a set of flow-paths witnessing that the total workload cost is at most $\kappa$.

**Claim 4.2.1.** It is necessarily the case that every directed edge $(u, v) \in E''$ is such that $(u, v)$ is a dynamic link in $H(N)$.

*Proof.* Suppose, for contradiction, that there is a directed edge $(u, v) \in E''$ so that $(u, v)$ is not a dynamic link in $N$. By construction, the node-to-node workload of $\alpha$ from $u$ to $v$ contributes a cost of at least $2\mu\alpha$ to the total workload cost (note that the cost of a static link is 1 and $2\mu < 1$ when $n \geq 3$). So, the total workload cost from flow-paths corresponding to directed edges of $E''$ is at least $\mu\alpha(|E''| + 1)$. However, $\kappa = \mu\alpha|E''| + 3\mu n(\lceil \log_2(n) \rceil + 3)$ and so we must have that $\mu\alpha = 4\mu n \log_2(n) \leq 3\mu n(\lceil \log_2(n) \rceil + 3)$ which yields a contradiction when $n \geq 2^{10}$. The claim follows. □

By Claim 4.2.1, the total workload cost of flow-paths corresponding to the directed edges of $E''$ is exactly $\mu\alpha|E''|$. Consequently, the total workload cost of flow-paths corresponding to the directed edges of $E^{root}$ must be at most $3\mu n(\lceil\log_2(n)\rceil + 3)$.

Given the set of dynamic links as supplied by Claim 4.2.1, the only other possible dynamic links we might have involve the leaf nodes $\{l_i : 1 \leq i \leq n\}$, the nodes of $\{v_i : 1 \leq i \leq 3n\}$ and nodes of $U$. If we are looking for a path of dynamic links joining $r$ and any node $x_i$ then no matter how we might extend the set of dynamic links corresponding to the directed edges of $E''$, we can never obtain a path of (graph-theoretic) length less than $d + 3$: a path of length $d$ down the tree $T$ from $r$ to a leaf $l$ followed by a path of length 3 of the form $(l, v_j, u_j, x_i)$ or $(l, v_j, w_j, x_i)$.

We may assume $U$ is not involved in any such path. Any path visiting some node $\bar{u} \in U$ can be shortened to a path of the former form by simply removing $\bar{u}$ from the path. This corresponds to a rewiring of the configuration $N$ which decreases the total workload cost so, for example, the path $(r, \ldots, l, \bar{u}, v_j, u_j, x_i)$ of length $n + 4$ can be shortened to $(r, \ldots, l, v_j, u_j, x_i)$ of length $n + 3$ by removing the dynamic links $(l, \bar{u})$ and $(\bar{u}, v_j)$ from $N$ and adding the dynamic link $(l, v_j)$ to $N$. Clearly this does not increase the workload cost for any demand.

So, any flow-path corresponding to some directed edge $(r, x_i) \in E^{root}$ and consisting entirely of dynamic links has workload cost at least $\mu(d + 3) = \mu(\lceil\log_2(n)\rceil + 3)$. If a flow-path corresponding to some directed edge $(r, x_i) \in E^{root}$ contains a static link then the workload cost of this flow-path is at least 1 and $1 > \frac{1}{2} + \frac{3}{2\lceil\log_2(n)\rceil} = \mu(\lceil\log_2(n)\rceil + 3)$, when $n \geq 2^4$. As there are $3n$ flow-paths corresponding to directed edges of $E^{root}$, we must have that every flow-path corresponding to a directed edge of $E^{root}$ consists entirely of dynamic links and has (graph-theoretic) length $\lceil\log_2(n)\rceil + 3$.

Consider the flow-path corresponding to the directed edge $(r, x_i) \in E^{root}$. Suppose that the dynamic link $(l_k, v_j)$ appears on this flow-path (exactly one such dynamic link does). Note that there are at most $n$ such dynamic links used in the flow-paths of $F$ corresponding to the directed edges of $E^{root}$. Build the set of clauses $\mathcal{C}' \subseteq \mathcal{C}$ by including every such $c_j$ in $\mathcal{C}'$. Consequently, we have a subset $\mathcal{C}' \subseteq \mathcal{C}$ of at most $n$ clauses. Moreover, as we can reach any $x_i$ from $r$ by a flow-path of $d$+3 dynamic links, every $x_i \in \mathcal{X}$ appears in a clause of $\mathcal{C}'$; that is, $|\mathcal{C}'| = n$, $\mathcal{C}'$ is an exact cover for $\mathcal{X}$ and $(\mathcal{X}, \mathcal{C})$ is a yes-instance of RXC3. As our instance $(H, \mu, D, \kappa)$ can be constructed from $(\mathcal{X}, \mathcal{C})$ in time polynomial in $n$, our result follows. □

These results led us to consider the problem 1-SWITCHED RRP$(\sigma = k)$ for $k \geq 0$. By extending Lemma 4.3 due to [38], we show that this restriction leads to a tractable problem when either $\sigma = 0$ or the static network is a complete graph, in contrast with our NP-completeness results. That result makes use of the *dynamic link limit* $\delta$ - recall that when $\delta = 1$ every flow-path must contain at most one dynamic link. In combination with the constraint $\sigma = 0$, this entails that every flow path consists of either a single dynamic link (and no static links), or of one or more static links (and no dynamic links). The result below is proven via a maximum matching argument in [38].

**Lemma 4.3** ([38, Theorem 3.1])**.** The problem RRP($\sigma = 0$, $\delta = 1$) can be solved in polynomial-time when there is only one switch. Moreover, the problem remains solvable in polynomial-time even if we allow non-uniform weights for dynamic links.

**Theorem 4.4.** 1-SWITCHED RRP($\sigma = 0$) is in P.

*Proof.* Note that this proof holds even if we allow variable weights for static and optic links. Because each node $v$ is connected to the switch exactly once, it is not possible that any vertex is incident to two optic links, and hence impossible for there to be any path consisting of two or more optic links. Consequently setting $\delta = 1$ introduces no new constraints; in this scenario a flow is permissible with $\sigma = 0$ if and only if it permissible with $\sigma = 0$ and $\delta = 1$. Hence the input instance of 1-SWITCHED RRP($\sigma = 0, \delta = 1$) is a yes-instance if and only the corresponding instance of RRP($\sigma = 0, \delta = 1$) is a yes-instance, and tractability follows from Lemma 4.3. □

**Corollary 4.5.** 1-SWITCHED RRP($\sigma = k$) restricted to instances where the static network $G$ is a complete graph is in P, for any $k \in \mathbb{N} \cup \{\infty\}$.

*Proof.* If $G$ is a complete graph with all edge weights equal (without loss take $wt(u, v) = 1\ \forall (u, v) \in E$) then without loss under any configuration $N$, each demand $D[u, v]$ is routed via the flow-path $\varphi(u, v)$ of minimum weight, which is either:

- a single static link from $u$ to $v$ with unit weight.
- a single dynamic link from $u$ to $v$ with weight $\mu$.

It follows that setting $\sigma = 0$ introduces no new constraints, and then by Theorem 4.4 we have tractability. □

Corollary 4.5 rules out the possibility that 1-SWITCHED RRP might be NP-complete for any polynomial graph family $\mathcal{H}$ (since such a claim would extend to the family of complete graphs) unless P equals NP. This leaves open the practically relevant case where $\Delta_S = 1$ and $\sigma > 0$ for *specific* topologies. We consequently consider the scenario where the static network is a hypercube and the segregation parameter $\sigma = 3$.

**Theorem 4.6.** For any fixed $\mu \in (0, 1)$, the problem 1-SWITCHED RRP($\sigma = 3$) restricted to instances $(H, \mu, D, \kappa)$ satisfying:

- $H \in \mathcal{Q}$, where $\mathcal{Q} := \{Q_d | d \in \mathbb{N}\}$ is the family of hypercubes
- the workload matrix $D$ is sparse and all values in it are polynomial in $n$

is NP-complete.

*Proof.* Note that RRP as in the statement of the theorem is in NP on account of the numerical restrictions imposed. As in the proof of Theorem 4.2, we reduce from the problem RXC3. Let $(\mathcal{X}, \mathcal{C})$ be an instance of RXC3 of size $n$. W.l.o.g., we may assume that $n$ is even. Let $m = \lceil \log_2(3n) \rceil$ (so, $3n \leq 2^m$). Our static network $H$

is the hypercube $Q_{8m}$ (so, $|Q_{8m}| = 2^{8m} = O(n^8)$) and there is exactly 1 switch link from every node to our switch $s$.

We begin by defining a weighted digraph $D' = (V, E')$ where $V$ is the node set of our hypercube $Q_{8m}$. The digraph $D'$ is to provide a description of our workload matrix. In order to define the directed edges of $D'$, we define some specific subsets of nodes of $V$. We explain the notation that we use as we proceed. By 'distance' we mean the distance (that is, shortest path length) in the hypercube $Q_{8m}$ and by $V_i$ (resp. $V_{\leq i}$) we mean the subset of nodes of $V$ that are at distance exactly $i$ (resp. at most $i$) from the *root node* $r = (1, 1, \ldots, 1) = 1^{8m}$ (in general, we also denote nodes of $V$ as bit-strings of length $8m$).

- Within the nodes of $V_m$, we define the subset $\tilde{P} = \{z\bar{z}1^{6m} : z \in \{0,1\}^m\}$ (in general: $yz$ denotes the concatenation of two bit-strings $y$ and $z$, with $z^i$ denoting the concatenation of $i$ copies of the bit-string $z$; and $\bar{z}$ denotes the complement of the bit-string $z$). Note that $|\tilde{P}| = 2^m$.
- Within the nodes of $V_{3m}$, we define the subset $\tilde{C} = \{(z\bar{z})^3 1^{2m} : z \in \{0,1\}^m\}$. Note that $|\tilde{C}| = 2^m$.
  - For any bit-string $z \in \{0,1\}^{8m}$ and for any bit-string $y \in \{0,1\}^i$ where $i < 8m$, we write $\oplus(z; y)$ to denote the bitwise exclusive OR of $z$ and $0^{8m-i}y$ (so, $\oplus(z; y)$ differs from $z$ on at most the last $i$ bits). For any set $U$ of bit-strings of length $8m$ and bit-string $y$ of length $i < 8m$, we define $U^{\oplus y} = \{\oplus(z; y) : z \in U\}$.
  - For any $c \in \tilde{C}$, we define
    * $c^{\oplus 001} = \oplus(c; 001)$
    * $c^{\oplus 010} = \oplus(c; 010)$
    * $c^{\oplus 100} = \oplus(c; 100)$

    and so we obtain sets of nodes $\tilde{C}^{\oplus 001}$, $\tilde{C}^{\oplus 010}$ and $\tilde{C}^{\oplus 100}$, each consisting of sets of neighbours of nodes in $\tilde{C}$ (note also that if $c_1, c_2 \in \tilde{C}$ are distinct then $c_1$ and $c_2$ are at distance at least 6 in $Q_{8m}$ and any node of $\{c_1^{\oplus 001}, c_1^{\oplus 010}, c_1^{\oplus 100}\}$ and node of $\{c_2^{\oplus 001}, c_2^{\oplus 010}, c_2^{\oplus 100}\}$ are at distance at least 6 in $Q_{8m}$).
- Within the nodes of $V_{4m}$, we define the subset $\tilde{X} = \{(z\bar{z})^4 : z \in \{0,1\}^m\}$. Note that $|\tilde{X}| = 2^m$.
  - For any $x \in \tilde{X}$, we define
    * $x^{\oplus 01} = \oplus(x; 01)$
    * $x^{\oplus 10} = \oplus(x; 10)$

    and so we obtain sets of nodes $\tilde{X}^{\oplus 01}$ and $\tilde{X}^{\oplus 10}$, each consisting of sets of neighbours of nodes in $\tilde{X}$ (note that if $x_1, x_2 \in \tilde{X}$ are distinct then $x_1$ and $x_2$ are at distance at least 8 in $Q_{8m}$ and any node of $\{x_1^{\oplus 01}, x_1^{\oplus 10}\}$ and node of $\{x_2^{\oplus 01}, x_2^{\oplus 10}\}$ are at distance at least 6 in $Q_{8m}$).

We define the weighted directed edges $E'$ of $D'$ as follows. There are three weights: some $\beta > \max\{\frac{3n(m+1+2\mu)}{\mu}, \frac{3n(m+1+2\mu)}{1-\mu}\}$; some $\alpha > 15n\beta + 9n^2 + 9n$; and 1. The directed edges of $E_\beta$ (resp. $E_\alpha$, $E_1$) all have weight $\beta$ (resp. $\alpha$, 1) and $E' = E_\beta \cup E_\alpha \cup E_1$.

- The directed edges of $E_\beta$ are derived using the structure of our instance $(\mathcal{X}, \mathcal{C})$ of RXC3.
  - For each $1 \leq j \leq 3n$, identify the clause $c_j \in \mathcal{C}$ with the node $c_j = (bin_m(j-1)\overline{bin_m(j-1)})^3 1^{2m}$ of the subset $\tilde{C}$ of $V_{3m}$, where in general $bin_i(j)$ is the binary representation of the natural number $j$ as a bit-string of length $i$ (so, in particular, $0 \leq j < 2^i$). Henceforth, we refer to these specific nodes of $\tilde{C}$ as the set of *clause nodes* $C = \{c_j : 1 \leq j \leq 3n\}$.
  - For each $1 \leq i \leq 3n$, identify the element $x_i$ of $\mathcal{X}$ with the node $x_i = (bin_m(i-1)\overline{bin_m(i-1)})^4$ of the subset $\tilde{X}$ of $V_{4m}$. Henceforth, we refer to these specific nodes of $\tilde{X}$ as the set of *element nodes* $X = \{x_i : 1 \leq i \leq 3n\}$.

  For any clause $c_j = \{x_1^j, x_2^j, x_3^j\}$ of $\mathcal{C}$, there are directed edges $(c_j^{\oplus 001}, x_1^i)$, $(c_j^{\oplus 010}, x_2^i)$ and $(c_j^{\oplus 100}, x_3^i)$ in $E_\beta$ and we refer to the nodes $c_j^{\oplus 001}$, $c_j^{\oplus 010}$ and $c_j^{\oplus 100}$ as the *associate clause nodes* of clause $c_j$ or of clause node $c_j$. We define the *associate element nodes* of an element or an element node analogously. This constitutes all directed edges of $E_\beta$.

- We denote by $P = \{p_i : 1 \leq i \leq n\}$ a subset of $n$ nodes of $\tilde{P}$ which we refer to as the set of *port nodes*. The directed edges of $E_\alpha$ are $E_\alpha^P \cup E_\alpha^W$ where:
  - $E_\alpha^P = \{(z, \bar{z}) : z \in V_{\leq m+5} \setminus P\}$
  - $E_\alpha^W$ is an arbitrary orientation of a specific perfect matching $M$ (coming up) on the nodes of $W \setminus (X \cup X^{\oplus 01} \cup X^{\oplus 10})$ where $W$ is defined as $V_{4m} \cup \bigcup_{i=1}^3 V_{4m-i} \cup \bigcup_{i=1}^3 V_{4m+i}$.

  Note that $|W \setminus (X \cup X^{\oplus 01} \cup X^{\oplus 10})|$ is even and so a perfect matching exists (recall, we assumed that $n$ is even); however, as stated, we require that our perfect matching $M$ has a specific property which we describe below.

- The set of directed edges $E_1$ is defined as $\{(r, x_i) : 1 \leq i \leq 3n\}$; that is, as $\{r\} \times X$.

Now for our perfect matching $M$.

**Claim 4.6.1.** There is a perfect matching $M$ of $W \setminus (X \cup X^{\oplus 01} \cup X^{\oplus 10})$ so that if $(u, v)$ is in the matching then there is no static link joining $u$ and $v$.

*Proof.* Define a perfect matching $M$ on $W$ as follows. First, match every node in $V_{4m}$ with the node $v$ that differs from $u$ in every bit; that is, $v = \bar{u}$.

Consider some node $z\bar{z}z\bar{z}z\bar{z}z\bar{z}$ of $X$. Its matched node is $\bar{z}z\bar{z}z\bar{z}z\bar{z}z$ which is either in $X$ or outside $X$. Suppose it is the latter. Suppose also that some other node $w\bar{w}w\bar{w}w\bar{w}w\bar{w}$ is in $X$ and its matched node $\bar{w}w\bar{w}w\bar{w}w\bar{w}w$ lies outside $X$. Note that the distance between the two matched nodes $\bar{z}z\bar{z}z\bar{z}z\bar{z}z$ and $\bar{w}w\bar{w}w\bar{w}w\bar{w}w$ is at least 8. We amend our matching $M$ by removing all matched pairs of nodes of $X$ and choose an arbitrary matching on the remaining matched nodes (note that there is an even number of such matched nodes). Any pair of nodes in $M$ is such that there is no static link joining them and every node of $V_{4m} \setminus X$ is involved in $M$.

Extend $M$ with a matching so that every node of $V_{4m-1} \setminus (X^{\oplus 01} \cup X^{\oplus 10})$ is matched with a node in $V_{4m+1} \setminus (X^{\oplus 01} \cup X^{\oplus 10})$. This is possible as $|V_{4m-1}| = |V_{4m+1}|$ and $|V_{4m-1} \cap (X^{\oplus 01} \cup X^{\oplus 10})| = |V_{4m+1} \cap (X^{\oplus 01} \cup X^{\oplus 10})|$. Further extend $M$ with a matching so that: every node of $V_{4m-2}$ (resp. $V_{4m-3}$) is matched with a node in $V_{4m+2}$ (resp. $V_{4m+3}$). The claim follows. □

The digraph $D'$ can be visualized in Figure 4.7. The rows are intended to illustrate nodes in different $V_i$'s, although the rows labelled $V_{3m-1/3m+1}$ and $V_{4m-1/4m+1}$ contain all nodes of $V_{3m-1} \cup V_{3m+1}$ and $V_{4m-1} \cup V_{4m+1}$, respectively (this is where the associate clause nodes and the associate element nodes lie, respectively). We have not shown all directed edges; just enough to give a flavour of the construction. Directed edges from $E_\beta$ are shown as solid directed edges with the directed edges corresponding to the clause $c_1 = \{x_1, x_n, x_2\}$, for example, depicted (the grey nodes are the associate clause nodes in order $c_1^{\oplus 001}, c_1^{\oplus 010}, c_1^{\oplus 100}, c_2^{\oplus 001}, \ldots$ from left to right). Directed edges from $E_\alpha$ are shown as dotted directed edges and the grey rectangle contains all nodes from $W$ involved in directed edges of $E_\alpha$. The white nodes within this rectangle are the element nodes and the associate element nodes (that is, the nodes of $W$ that are not incident with directed edges from $E_\alpha$). Finally, the dashed directed edges depict the directed edges from $E_1$ and the port nodes and clause nodes are also shown in white.

In order to complete our instance $(H, \mu, D, \kappa)$, we define $\mu$ to be any fixed (rational) value in the interval $(0, 1)$ and $\kappa = \kappa_\beta + \kappa_\alpha + \kappa_1$ where:

- $\kappa_\beta = \frac{|E_\beta|}{3}(\mu\beta) + \frac{2|E_\beta|}{3}((\mu+1)\beta) = 3n\mu\beta + 6n(\mu+1)\beta$
- $\kappa_\alpha = |E_\alpha|\mu\alpha$
- $\kappa_1 = |E_1|(m + 1 + 2\mu) = 3n(m + 1 + 2\mu)$.

Suppose that there is an exact cover $\mathcal{C}' = \{c_{j_i} : 1 \leq i \leq n\} \subseteq \mathcal{C}$ of $\mathcal{X}$. Let the set of clause nodes $C' \subseteq C$ be $\{c_{j_i} : 1 \leq i \leq n\}$. We define a configuration $N$ by choosing dynamic links as follows.

- For some arbitrary bijection $f$ from $P$ to $C'$, $\{(p, f(p)) : p \in P\}$ is a set of $n$ dynamic links.
- For each clause $c_j = \{x_1^j, x_2^j, x_3^j\} \in \mathcal{C}'$, there are dynamic links $(c_j^{\oplus 001}, x_1^j)$, $(c_j^{\oplus 010}, x_2^j)$ and $(c_j^{\oplus 100}, x_3^j)$; so, all nodes of $X$ are incident with a dynamic link as are all associate clause nodes of $\{c_j^{\oplus 001}, c_j^{\oplus 010}, c_j^{\oplus 100} : c_j \in C'\}$. Note that these dynamic links result from the directed edges of $E_\beta$ incident with the associate clause nodes of the clause nodes of $C'$.

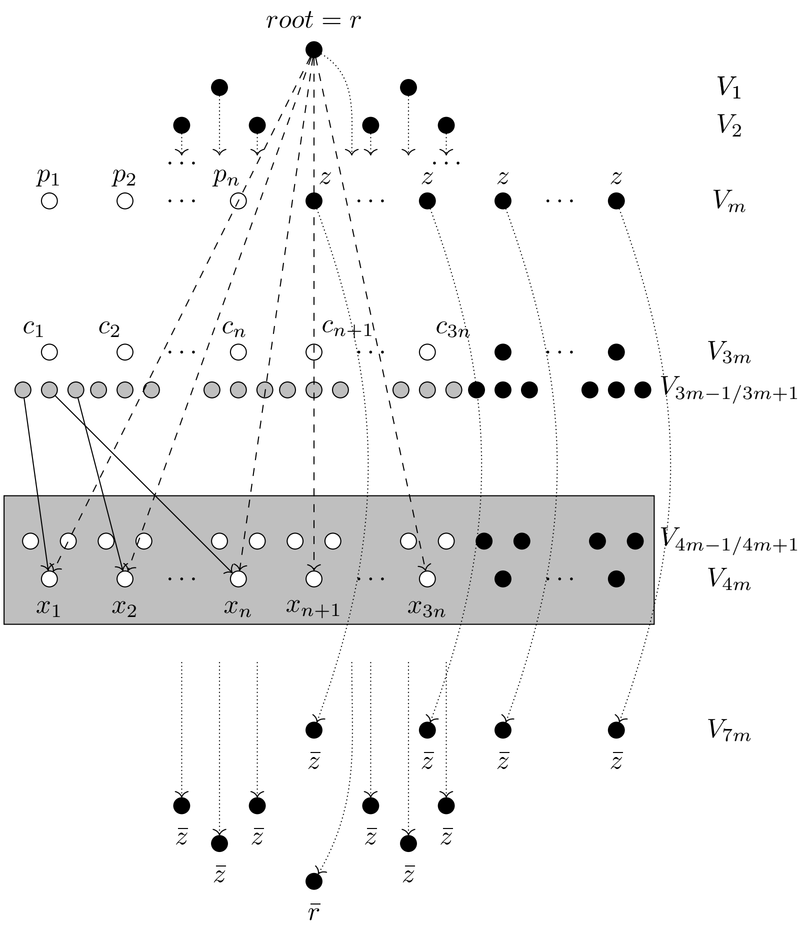


Figure 4.7: The graph $D'$.

- All directed edges of $E_\alpha$ result in dynamic links. In particular, there is no dynamic link from a node outside $W$ to a node inside $W$ except possibly incident with the element nodes and the associate element nodes; indeed, the element nodes are already incident with such 'external' dynamic links.

- Consider the node $x_1 \in X$. The element $x_1$ appears in two clauses of $\mathcal{C} \setminus \mathcal{C}'$, say $c_{i_1}$ and $c_{i_2}$. Suppose, for example, that the element $x_1$ appears as the second element of clause $c_{i_1}$ and as the first element of clause $c_{i_2}$. If so then include a dynamic link from each of $c_{i_1}^{\oplus 010}$ and $c_{i_2}^{\oplus 001}$ to $x_1^{\oplus 01}$ and $x_1^{\oplus 10}$. Alternatively, if, say, the element $x_1$ appears as the third element of clause $c_{j_1}$ and as the third element of clause $c_{j_2}$ then include a dynamic link from each of $c_{j_1}^{\oplus 100}$ and $c_{j_2}^{\oplus 100}$ to $x_1^{\oplus 01}$ and $x_1^{\oplus 10}$. Proceed similarly and analogously with all remaining element nodes of $X \setminus \{x_1\}$. On completion of this iterative process, there is a dynamic link from every associate clause node to a unique element node or associate element node; that is, these dynamic links depict a bijection from the associate clause nodes to the element nodes and the associate element nodes.

This constitutes the configuration $N$.

Consider some $\beta$-demand of our workload. For some node $c_j \in C'$, all such workloads originating at the nodes $c_j^{\oplus 001}$, $c_j^{\oplus 010}$ and $c_j^{\oplus 100}$ can be served via a flow-path consisting of a solitary dynamic link at workload cost $\mu\beta$. For some node $c_j \in C \setminus C'$, suppose that there is a demand $D[c_j^{\oplus 001}, x_1^i] = \beta$. This demand exists because the element $x_i$ is the first element of clause $c_j$. As the element $x_i$ is the first element of clause $c_j$, there is a dynamic link from $c_j^{\oplus 001}$ to a neighbour of $x_i$ in $\{x_i^{\oplus 01}, x_i^{\oplus 10}\}$, which w.l.o.g. we may assume to be $x_i^{\oplus 01}$; hence, the demand can be served via the flow-path $c_j^{\oplus 001}, x_i^{\oplus 01}, x_i$ at workload cost $(\mu+1)\beta$. Defining other flow-paths analogously means that we can find flow-paths corresponding to the directed edges of $E_\beta$ the total workload cost of which is $3n\mu\beta + 6n(\mu+1)\beta = \kappa_\beta$.

All $\alpha$-demands can be served via a flow-path consisting of a solitary dynamic link at total workload cost $|E_\alpha|\mu\alpha = \kappa_\alpha$.

Consider some 1-demand $D[r, x_i] = 1$. Let the clause $c_j \in \mathcal{C}'$ be such that $x_i$ is an element of $c_j$; hence, there is a dynamic link from some port node $p_{j'}$ of $P$ to $c_j$ and a dynamic link from some neighbour of $c_j$ from $\{c_j^{\oplus 001}, c^{\oplus 010}, c^{\oplus 110}\}$ to $x_i$. Consequently, there is a static path from $r$ to $p_{j'}$ of weight $m$, a dynamic link from $p_{j'}$ to the node $c_j$, a static link from $c_j$ to the above neighbour in $\{c_j^{\oplus 001}, c^{\oplus 010}, c^{\oplus 110}\}$ and a dynamic link from this neighbour to $x_i$. Hence, the demand $D[r, x_i] = 1$ can be served via a flow-path of workload cost $m + 1 + 2\mu$ with all 1-demands served via flow-paths at a total workload cost $3n(m + 1 + 2\mu) = \kappa_1$. Consequently, our instance $(H, \mu, D, \kappa)$ is a yes-instance of 1-SWITCHED RRP$(\sigma = 3)$.

Conversely, suppose it is the case that $(H, \mu, D, \kappa)$ is a yes-instance of 1-SWITCHED RRP$(\sigma = 3)$ and that $N$ is a configuration and $F$ a set of flow-paths witnessing that the total workload cost is at most $\kappa$.

**Claim 4.6.2.** Every directed edge $(u, v)$ of $E_\alpha$ is necessarily such that $(u, v) \in N$ and the total workload cost of flow-paths serving the $\alpha$-demands is exactly $\kappa_\alpha$.

*Proof.* Suppose that at least one of the $\alpha$-demands is served via a flow-path at a workload cost of more than $\mu\alpha$; so, it must be at a workload cost of more than $(1 + \mu)\alpha$ as we cannot traverse a dynamic link followed immediately by another dynamic link (recall, $\Delta_S = 1$) and by Claim 4.6.1, if $D[u, v] = \alpha$ then there is no static link $(u, v)$. Hence, the total workload cost of flow-paths serving the $\alpha$-demands is at least $(|E_\alpha| - 1)\mu\alpha + (1 + \mu)\alpha = \kappa_\alpha + \alpha$. We have that $\kappa_\beta + \kappa_1 = 3n\mu\beta + 6n(\mu+1)\beta + 3n(m+1+2\mu) < 3n\beta + 12n\beta + 3n(3n+3) = 15n\beta + 9n^2 + 9n < \alpha$. Hence, the total workload cost of all flow-paths of $F$ is strictly greater than $\kappa$ which yields a contradiction and the claim follows. □

Call the dynamic links corresponding to the directed edges of $E_\alpha$ the *$\alpha$-dynamic links*.

**Claim 4.6.3.** Exactly $3n$ (resp. $6n$) $\beta$-demands are served by a flow-path of workload cost $\mu\beta$ (resp. $(1 + \mu)\beta$) exactly; that is, exactly $3n$ (resp. $6n$) flow-paths serving the $\beta$-demands consist of a dynamic link (resp. a dynamic link and a static link). Hence, the total workload cost of the flow-paths serving $\beta$-demands is exactly $\kappa_\beta$.

*Proof.* All $\beta$-demands are from a unique associate clause node to an element node. As $\Delta_S = 1$, at most $3n$ dynamic links can be incident with an element node. So, at most $3n$ $\beta$-demands are served by a flow-path consisting of a solitary dynamic link.

If a $\beta$-demand is served by a flow-path that does not consist of a solitary dynamic link then the flow-path has workload cost at least $(1+\mu)\beta$.

Suppose there are less than $3n$ $\beta$-demands that are served by a flow-path consisting of a solitary dynamic link. So, the total workload cost of flow-paths serving $\beta$-demands is at least $(3n-1)\mu\beta + (6n+1)(1+\mu)\beta = 9n\mu\beta + 6n\beta + \beta$. By Claim 4.6.2, $9n\mu\beta + 6n\beta + \beta \leq \kappa_\beta + \kappa_1 = 3n\mu\beta + 6n(\mu+1)\beta + 3n(m+1+2\mu)$; that is, $\beta \leq 3n(m+1+2\mu)$ which yields a contradiction. So, there are exactly $3n$ $\beta$-demands served by a flow-path consisting of a solitary dynamic link.

Suppose that there is a $\beta$-demand served by a flow-path of workload cost neither $\mu\beta$ nor $(1+\mu)\beta$. Such a flow-path has workload cost at least $\gamma\beta$ where $\gamma = \min\{2, 1+2\mu\}$. As before, the total workload cost of flow-paths serving $\beta$-demands is at least $3n\mu\beta + (6n-1)(1+\mu)\beta + \gamma\beta = 9n\mu\beta + 6n\beta + (\gamma - 1 - \mu)\beta$. By Claim 4.6.2, $9n\mu\beta + 6n\beta + (\gamma-1-\mu)\beta \leq \kappa_\beta + \kappa_1 = 3n\mu\beta + 6n(\mu+1)\beta + 3n(m+1+2\mu)$; that is, $(\gamma-1-\mu)\beta \leq 3n(m+1+2\mu)$. If $\gamma = 2$ then $\beta \leq \frac{3n(m+1+2\mu)}{1-\mu}$; and if $\gamma = 1+2\mu$ then $\beta \leq \frac{3n(m+1+2\mu)}{\mu}$. Whichever is the case, we obtain a contradiction. Hence, there are exactly $6n$ $\beta$-demands served by a flow-path of workload cost $(1+\mu)\beta$. Each of these flow-paths consists of a dynamic link and a static link and the claim follows. □

By Claim 4.6.3, each element node $x$ is incident via a dynamic link to a unique associate clause node of some clause $c$ so that the element $x \in \mathcal{X}$ is in the clause $c \in \mathcal{C}$. Consider some $\beta$-demand served by a flow-path of weight $1+\mu$; that is, a flow-path $f$ consisting of a static link and a dynamic link. By Claim 4.6.2, every node of $W \setminus (X \cup X^{\oplus 01} \cup X^{\oplus 10})$ is incident with a dynamic link incident with some other node of $W \setminus (X \cup X^{\oplus 01} \cup X^{\oplus 10})$. So, no node of $W \setminus (X \cup X^{\oplus 01} \cup X^{\oplus 10})$ can appear on $f$. Also, no associate clause node is adjacent via a static link to any other associate clause node. Hence, $f$ must be of the form $c', x', x_i$, where $c'$ is an associate clause node, of some clause $c$, that is adjacent via a dynamic link to the associate element node $x'$ which is adjacent via a static link to the element node $x$ and where the element $x \in \mathcal{X}$ is in the clause $c \in \mathcal{C}$. In particular:

- for every $x_i \in \mathcal{X}$, if $x_i \in c_1^i, c_2^i, c_3^i$, with $c_1^i, c_2^i, c_3^i \in \mathcal{C}$, then there are 3 dynamic links incident with a unique node of $\{x_i, x_i^{\oplus 01}, x_i^{\oplus 10}\}$ and incident with exactly one node of each of the sets of associate clause nodes of $c_1^i$, $c_2^i$ and $c_3^i$
- if there is a dynamic link from an element node $x$ or one of its associate element nodes to an associate clauses node of some clause $c$ then the element $x \in \mathcal{X}$ is in the clause $c \in \mathcal{C}$.

Call these dynamic links the *$\beta$-dynamic links*. Also, by Claim 4.6.3, the total workload cost of the flow-paths serving the $\beta$-demands is $\kappa_\beta$. Our aim now is to show that there are $n$ clauses with the property that every associate clause node of any of these clauses is incident with a $\beta$-dynamic link incident with an element node. Doing so would result in our instance of RXC3 being a yes-instance.

Consider some flow-path $f$ that services some 1-demand. Our preliminary aim is to show that every such flow-path $f$ has weight at least $m+1+2\mu$.

Suppose that $f$ involves 2 dynamic links. Hence, the structure of $f$ is $s^*ds^+d$ or $ds^+ds^*$, where $s$ (resp. $d$) denotes a static (resp. dynamic) link and $*$ (resp. $+$) denotes at least 0 (resp. at least 1) occurrence (in a regular-language style); recall that $\sigma = 3$.

- If $f$ has structure $s^*ds^+d$ then the second dynamic link must be of the form $(c', x)$, where $c'$ is an associate clause node and $x$ is an element node. If the first dynamic link is not an $\alpha$- or $\beta$-dynamic link then the initial prefix of static links has weight at least $m$. Hence, the total weight of $f$ is at least $m + 1 + 2\mu$. Alternatively, suppose that the first dynamic link is an $\alpha$- or $\beta$-dynamic link.
    - If it is an $\alpha$-dynamic link and is of the form $(z, \bar{z})$, for some $z \in V_{\leq m} \setminus P$, then the total weight of $f$ is at least $i + \mu + 5m - i - 1 + \mu$, for some $0 \leq i \leq m$; that is, at least $5m - 1 + 2\mu > m + 1 + 2\mu$.
    - If it is an $\alpha$-dynamic link and is of the form $(u, v)$, for some $u, w \in W$, then the total weight of $f$ is at least $4m - 1 + \mu + 1 + \mu = 4m + 2\mu > m + 1 + 2\mu$.
    - If it is a $\beta$-dynamic link then the total weight of $f$ is at least $3m - 1 + \mu + 1 + \mu = 3m + 2\mu > m + 1 + 2\mu$.
- If $f$ has structure $ds^+ds^*$ then the first dynamic link must be $(r, \bar{r})$.
    - Suppose that the first sub-path of static links has weight less than or equal to $m$ with the second dynamic link being an $\alpha$-dynamic link of the form $(\bar{z}, z)$. Then the total weight of $f$ is at least $\mu + i + \mu + 4m - i$, for some $0 \leq i < m$; that is, $4m + 2\mu > m + 1 + 2\mu$.
    - Suppose that the first sub-path of static links has weight exactly $m$ with the second dynamic link not an $\alpha$- or $\beta$-dynamic link. As all nodes of $W$ are already incident with a dynamic link, we must have that the weight of $f$ is at least $\mu + m + \mu + 4 = m + 4 + 2\mu > m + 1 + 2\mu$.
    - Suppose that the first sub-path of static links has weight greater than $m$. The weight of $f$ must be at least $\mu + m + 1 + \mu + 1 = m + 2 + 2\mu > m + 1 + 2\mu$.

Suppose that $f$ involves 1 dynamic link; so, the structure of $f$ is $s^*ds^*$. If the dynamic link is an $\alpha$- or $\beta$-dynamic link then no matter which dynamic link this is, the weight of $f$ is greater than $3m - 1 + \mu > m + 1 + 2\mu$. If the dynamic link is not an $\alpha$- or $\beta$-dynamic link then the weight of $f$ is at least $m + \mu + 4 > m + 1 + 2\mu$, as this dynamic link cannot be incident with any node of $W$. Finally, if $f$ does not involve a dynamic link then the weight of $f$ is at least $4m > m + 1 + 2\mu$. Hence, every flow-path serving some 1-demand has weight at least $m + 1 + 2\mu$ and when it has weight exactly $m + 1 + 2\mu$, it takes the form of a static path from $r$ to some port node of $P$, augmented with a dynamic link to some neighbour of an associate clause node, augmented with a static link to the associate clause node, and augmented with a dynamic link from the associate clause node to some element node. As the total workload cost of all flow-paths serving 1-demands is at most $\kappa_1 = 3n(m + 1 + 2\mu)$, every such flow-path has weight exactly $m + 1 + 2\mu$ and is therefore of the form just described.

There are $3n$ 1-demands yet only $n$ dynamic links from a port node to some neighbour of an associate clause node. Hence, these $n$ neighbours of associate clause nodes must lie on $3n$ flow-paths. Consequently, these neighbours of associate clause nodes must actually be clause nodes. Let these clause nodes be $C' = \{c_{j_1}, c_{j_2}, \ldots, c_{j_n}\}$. Thus, we have that there is a dynamic link from every associate clause node of each $c_{j_i}$ to some element node and we have a subset of clauses $\mathcal{C}' = \{c_{j_1}, c_{j_2}, \ldots, c_{j_n}\}$ so that every element of $\mathcal{X}$ lies in exactly one clause of $\mathcal{C}'$; that is, $(\mathcal{X}, \mathcal{C})$ is a yes-instance of RXC3. The result follows as our instance $(H, \mu, D, \kappa)$ can clearly be constructed from $(\mathcal{X}, \mathcal{C})$ is time polynomial in $n$. □

We emphasize the relevance of the hypercube as a prototypical model of interconnection networks (see, e.g., [40]) and the fact that we obtain hardness here for any choice of fixed dynamic link weight $\mu$ between 0 and 1.

Taken together our results comprehensively establish the computational hardness of RRP in practically relevant settings. In particular, we establish that the problem remains intractable in several cases where the demand matrix is sparse, the hybrid network is highly structured (in fact node-symmetric) and the weights of links depend only on their medium.

### 4.3.1 The case of $\delta = 1$

This section is devoted to the restriction of RRP where the dynamic link limit $\delta$ is set to 1 - where any flow-path must use no more than a single dynamic link. We shall require substantially different techniques from those we have used up until now so to prove our main result in the remainder of the chapter, which entails that the problem 1-SWITCHED RRP$(\delta = 1)$ is NP-complete for various graph classes including hypercubes, grids and toroidal grids. The intention is that this section provides a convenient template for proving hardness of the problem for other practically interesting classes beyond the ones explicitly considered here.

#### Additional definitions

We begin with some basic definitions from graph theory. Let $G = (V, E)$ be a simple undirected graph. We define the *open neighborhood* of a vertex $v$ to be $N(v) := \{u : (u, v) \in E\}$, and its closed neighborhood $N[v] := N(v) \cup \{v\}$. Likewise for any set of vertices $S$, we define $N[S] := \cup_{v \in S} N[v]$ and $N(S) := N[V] \setminus S$. A *dominating set* in $G$ is a set of vertices $s$ such that each vertex in $G$ is either in $S$ or adjacent to a vertex in $S$. The *domination number* of $G$, denoted $\gamma(G)$, is the least number of vertices in any dominating set of $G$. DOMINATING SET is the decision problem asking, for input $G$ and $k$, whether $\gamma(G) \leq k$.

Where $S \subseteq V$ is a set of vertices in the graph, we denote $G[S]$ the subgraph of $G$ induced by $S$. That is, $G[S]$ has $S$ as its set of vertices and as edges exactly those edges of $G$ with both endpoints incident to a vertex in $S$. Also, $\text{dist}_G(u,v)$ is the number of edges on the shortest path between $u$ and $v$ in $G$.

**Definition 4.7** (Ball, Sphere)**.** Where $G$ is a graph, we denote $B_G(v,r)$ (resp. $S_G(v,r)$) the *ball* (resp. *sphere*) with center $v$ and radius $r$ in $G$. Formally these are sets of vertices defined as:

$$B_G(v,r) := \{u : \text{dist}_G(u,v) \leq r\}$$
$$S_G(v,r) := \{u : \text{dist}_G(u,v) = r\}$$

We shall also make use of the following (likely non-standard) definitions.

**Definition 4.8** (Restriction of decision problems to a graph class)**.** Let $\Pi$ be a decision problem which takes one or many inputs, exactly one of which is a simple undirected graph. We denote by $\Pi(\mathcal{G})$ the restriction of $\Pi$ to the graph class $\mathcal{G}$. That is, $\Pi(\mathcal{G})$ has as yes-instances (resp. no-instances) exactly those yes-instances (resp. no-instances) of $\Pi$ where the graph portion of the input belongs to $\mathcal{G}$.

We introduce a new graph problem, which is subtly different from the classic Dominating Set, and which has the property that it may remain hard even when the graph portion of the input is highly structured. This subtlety will become important presently - note that the property is precisely that which we wish to study for 1-Switched RRP($\delta = 1$).

Partial Domination
*Input:* Simple undirected graph $G = (V, E)$; set $T \subseteq V$; integer $k$.
*Question:* Is $(G[T], k)$ a yes-instance of Dominating Set? Equivalently, is there some set $X \subseteq T$ with $|X| \leq k$ such that $T \subseteq N_G[X]$?

### Hardness of Partial Domination on (toroidal) grids and hypercubes

We now show that Partial Domination is NP-complete for several graph classes of interest to us, namely grids and hypercubes. This computational hardness (together with another graph class property we shall come to later) provides the foundation for our proof of Theorem 4.19. We note that Dominating Set is trivially tractable for grids, though the same cannot be said of hypercubes; even $\gamma(Q_{10})$ is unknown [134]. This subsection leverages several results from the literature, which more or less straightforwardly yield the desired results.

**Theorem 4.9** ([135] Theorem 5.1)**.** Dominating Set(Induced subgraphs of grids) is NP-complete.

Note that the corollary below relies on some subtle properties of the proof applied in [135] (namely, that the construction provided explicitly describes an embedding into some grid). It is in general NP-hard, given some graph, to determine whether it is an induced subgraph of a grid (and also to produce its vertices' coordinates in a grid - see [136] and references therein).

**Corollary 4.10.** Partial Domination(Grids) is NP-complete.

We now turn to the other graph class of interest to us - the hypercubes $\mathcal{Q}$. Although it is straightforward to show that every grid is the induced subgraph of some hypercube, we require for technical reasons that, more strongly, the hypercube in question is at most polynomially larger than the contained grid. Fortunately, we are able to rely here on the extensive literature surrounding the so-called *snake-in-the-box* problem, which consists in finding large induced cycles in hypercubes. The following result belongs to that body of work.

**Theorem 4.11** (Abbott and Katchalski [137, 138])**.** For any $d \geq 2$, the hypercube $Q_d$ contains an induced path on $\frac{77}{256}2^d$ vertices. Given $d$, the coordinates of such a path may be produced in time polynomial in $2^d$.

In their work, Abbott and Katchalski state that there is an induced cycle on **strictly more than** $\frac{77}{256}2^d$ vertices, which entails an induced path on exactly $\frac{77}{256}2^d$ vertices. The authors do not discuss the runtime of their construction in their work. However, it is clear from their proof that their description of the induced cycle can be realized as a recursive algorithm with a runtime as stated in the theorem. Their result has the following consequence, which shall be useful to us towards proving the computational hardness of Partial Domination($\mathcal{Q}$).

**Corollary 4.12.** For any $d \geq 2$, the hypercube $Q_{2d}$ contains an induced grid on $(\frac{77}{256}2^d)^2$ vertices. Given $d$, the coordinates of such a grid may be produced in time polynomial in $2^d$.

*Proof.* Given $d$, produce coordinates of a path $P = \{p_1, p_2, \ldots, p_\ell\}$ of length $\ell = \frac{77}{256}2^d$ in $Q_d$ by applying Theorem 4.11. Note that each (node) $p_i$ is a bitstring of length $d$ exactly. Then the set $\{p_i p_j : i, j \in [\ell]\}$ is an induced grid of size $(\frac{77}{256}2^d)^2$ in $Q_{2d}$, and the result follows. □

It remains to combine the above results.

**Theorem 4.13.** Partial Domination($\mathcal{Q}$) is NP-complete.

*Proof.* Let $(G, T, k)$ be an instance of Partial Domination(Grids). We assume without loss of generality that $G$ is an $n \times n$ grid (if necessary, by extending $G$ in some dimension and retaining the same set $T$). We shall denote each vertex in $G$ by $v_{i,j}$ with $i, j \in [n]$. Let $d = \lceil \log_2(\frac{256n}{77}) \rceil$. Applying Corollary 4.12, we may efficiently produce a mapping from $V(G)$ (vertices of the $n \times n$ grid) to $V(Q_{2d})$. Denote this mapping $f$. Let $G' = Q_{2d}$, and $T' = \{f(t) : t \in T\}$. Then $G'[T'] = G[T]$ (so $\gamma(G'[T']) = \gamma(G[T])$ also) and $(G', T', k)$ is an instance of Partial Domination($\mathcal{Q}$). The construction of $G'$ and $T'$ is feasible in polynomial time, and the result follows. □

A reader interested in showing that some other graph class $\mathcal{G}$ (e.g. subcubic graphs) is hard for 1-Switched RRP may substitute the above for a proof that

Partial Domination($\mathcal{G}$) is NP-complete. As briefly alluded to earlier, this is one of two properties we shall require of a graph class in the proof of Theorem 4.19; we turn to the second property presently.

### Lunar graph classes

We have chosen to adopt a celestial metaphor to aid intuition, since in defining the class we make use of spheres, balls, large distances, and vast differences in size. The reader may find Figures 4.8 and 4.9 helpful in illustrating the definition.

**Definition 4.14** (Moon, Planet, Sun)**.** We say a graph $G$ is the *moon* in some graph $H$ if:

- There is some set $M \subset V(H)$ with $H[M] = G$. We also call $M$ the *moon* in $H$.
- There is some $o \in V(H)$ and integer $r$ such that the set $\tilde{P} := S(o, r)$ is a sphere of cardinality at least $|M|$ and $P := B(o, r)$ is the ball with the same center and radius. We call $P$ the *planet* and $\tilde{P}$ the *planet surface* (or, briefly, *surface*) in $H$.
- $N[M] \cap N[P] = \emptyset$ (the planet and the moon are far apart).
- $|\tilde{P}| \geq |M|$ (the surface is bigger than the moon).
- There is some set $S \subset V(H)$ (the *sun* in $H$) such that $|S| \geq |N[P]| + |N[M]|$ (the sun is bigger than the moon and planet together) and $N[S]$ is disjoint from $N[P]$ (the sun and planet are far apart). Moreover any node in $S$ is at distance at least $2r$ from any node in $M$ (the sun and moon are very far apart).

We continue with an astronomical theme for the naming of the graph class of interest itself:

**Definition 4.15** (lunar graph class)**.** We say a graph class $\mathcal{L}$ is *lunar* in a graph class $\mathcal{H}$ if, for each $G \in \mathcal{L}$, there exists some $H \in \mathcal{H}$ such that $G$ is the moon in $H$. We further require that such a graph $H$ (and vertex sets $M, \tilde{P}, S$ within it) can be constructed in polynomial time from $G$ (which entails that $H$ is at most polynomially larger than $G$). We say a graph class $\mathcal{L}$ is *lunar in itself* (or simply *lunar*) if the above holds for $\mathcal{H} = \mathcal{L}$.

For this definition to be useful, we still need to show that it holds for those graph classes which we are interested. The previously hypothesized reader interested in proving NP-completeness of 1-Switched RRP($\delta = 1$) restricted to, e.g., subcubic graphs, may find the following a useful blueprint to prove that subcubic graphs are lunar.

**Lemma 4.16.** The class of grid graphs is lunar in itself.

*Proof.* The reader may find the illustrative example in Figure 4.8 helpful. Let $G$ be some $n \times m$ grid (w.l.o.g. $n \geq m$, so $|G|$ is polynomial in $n$).

We shall use the fact that a sphere of radius $r \geq 1$ in a grid (which does not spill over the grid's boundary) has size $4r$.

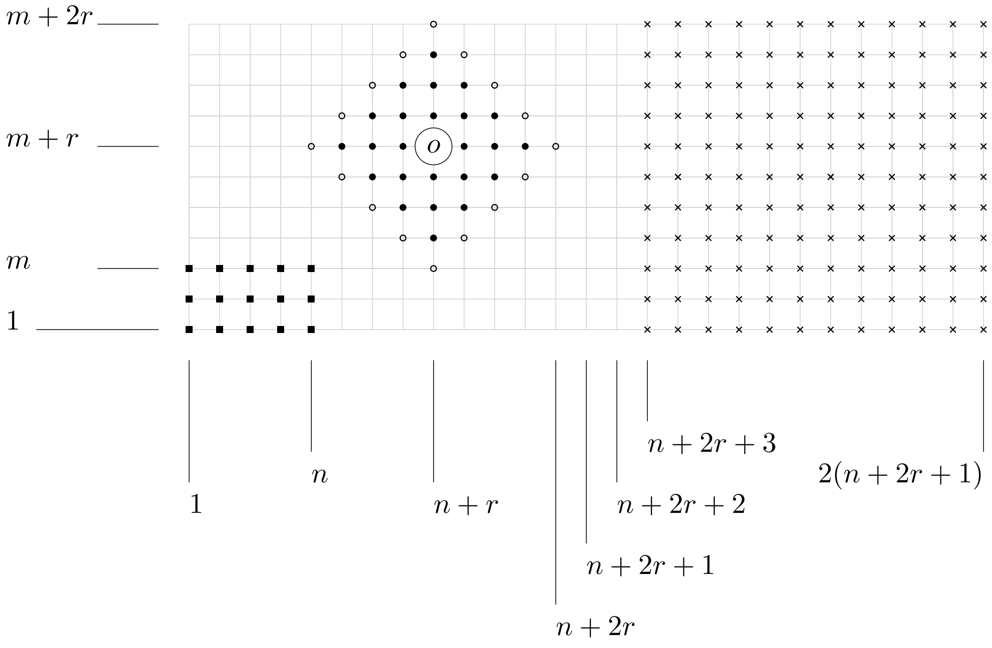


Figure 4.8: Illustration of our construction where $n = 5, m = 3, r = 4$. The $28 \times 11$ grid contains the $5 \times 3$ grid as a moon. Marked are: the moon $M$ (square vertices); $o = (n+r, m+r) = (9, 7)$, together with the planet $P = B(o, r)$ ($o$ and disk and circle vertices) and its surface $\tilde{P} = S(o, r)$ (circle vertices); the sun $S = \{(i, j) : i \geq 16\}$ (cross vertices).

Let $r = \lceil \frac{nm}{4} \rceil$. Let $x = 2(n + 2r + 1)$ and let $y = 2(m + 2r)$. We choose $H$ to be the $x \times y$ grid. Then we identify:

- The moon: $M = V(G)$ (vertices of the $n \times m$ grid are also vertices of $x \times y$ grid).
- The planet: $o = (n + r, m + r)$, $\tilde{P} = S(o, r)$, $P = B(o, r)$.
- The sun: $S = \{(i, j) : i \geq n + 2r + 3\}$.

It is easy to verify that the neighborhoods of the moon, planet and sun are disjoint; that the sun and moon are at distance at least $2r$; that the size of the planet exceeds that of the moon; and that the size of the sun exceeds that of the moon and planet's neighborhoods combined. Since our construction may be carried out in polynomial time, the result follows. $\square$

The $x \times y$ grid is an induced subgraph of the $2x \times 2y$ toroidal grid, with the notable property that every shortest path in the former remains a shortest path in the latter. Applying this fact, the proof above can straightforwardly be adapted to obtain the following corollary:

**Corollary 4.17.** The class of grid graphs is lunar in the class of toroidal grid graphs.

Note that the class of toroidal grid graphs is *not* lunar in itself. We now turn to the protagonist of this chapter, the class of hypercubes.

**Lemma 4.18.** The class of hypercubes $\mathcal{Q}$ is lunar in itself.

*Proof.* Note that we reuse some of the notation from our proof of Theorem 4.6. Let $G \in \mathcal{Q}$ be some hypercube of dimension $d$, i.e. $G = Q_d$ for some $d$. We define:

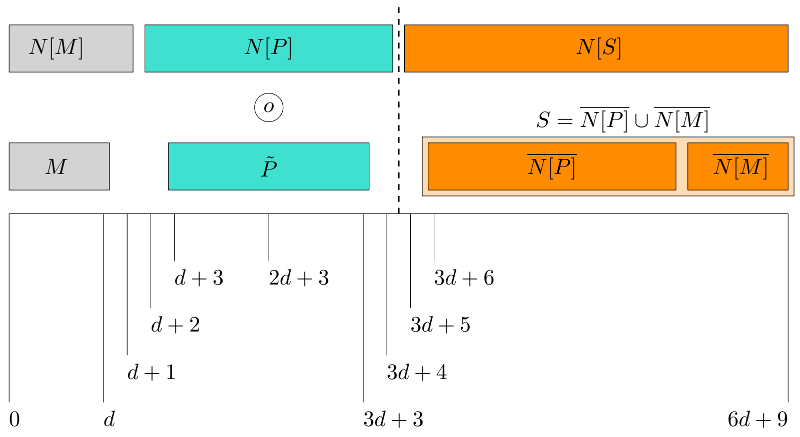


Figure 4.9: Illustration of our proof hypercubes are lunar, showing the weights (number of ones in any vertex label) for different sets of vertices. The midsection of the hypercube (separating majority-1 vertices from majority-0 vertices) is shown as a dashed line.

- Let $\ell = 6d+9$ and $H = Q_\ell$.
- Let $M := \{0^{5d+9}x : x \in Q_d\}$.
- Let $o := 0^{4d+6}1^{2d+3}$ and $r = d$. Recall we denote the planet $P = B(o,r)$ and its surface $\tilde{P} = S(o,r)$.
- Let $S := \{\overline{x} : x \in N[P] \cup N[M]\}$. Intuitively, the sun nodes are the reflection of the neighborhoods of planet nodes and moon nodes.

The illustration in Figure 4.9 will be helpful in verifying the following:

- $H[M] = G$.
- The sphere $\tilde{P} = S(o,r)$ has cardinality at least $|Q_d|$ (easy to see by considering $\{o \oplus 0^{4d+9}\overline{x}x : x \in Q_d\}$, which clearly contains only vertices at distance exactly $d$ from $o$ and so is a subset of $\tilde{P}$).
- $N[M]$ and $N[P]$ are disjoint.
- $N[M]$ and $N[P]$ contain only vertices with at most $3d+4$ ones.
- $|S| \geq |N[P]| + |N[M]|$ (by applying both two points above).
- $N[S]$ is disjoint from $N[M]$ and $N[P]$.
- Moreover any node in $S$ is at distance at least $2r$ from any node in $M$.

Note that $Q_\ell$ has size $2^{6d+9} = 2^9 \cdot (2^d)^6$ which is polynomial in the size of $Q_d$. The result follows.

□

### The main result

We are now able to state and prove the main result of this subsection.

**Theorem 4.19.** For any fixed $\mu \in (0,1)$, and any graph classes $\mathcal{L}$ and $\mathcal{G}$ with $\mathcal{L}$ lunar in $\mathcal{G}$, the problem PARTIAL DOMINATION($\mathcal{L}$) is polynomially reducible to the problem 1-SWITCHED RRP($\delta = 1$) restricted to instances $(H, \mu, D, \kappa)$ satisfying:

- $H \in \mathcal{G}$, and
- the workload matrix $D$ is sparse and all values in it are polynomial in $|H|$.

*Proof.* We are given an instance $(G, T, k)$ of Partial Domination($\mathcal{L}$), with $G \in \mathcal{L}$, a set of vertices $T \subseteq V(G)$, and integer $k$. We shall produce an instance $(H, \mu, D, \kappa)$ of 1-Switched RRP($\delta = 1$). $\mu$ is some prescribed value between 0 and 1, as in the theorem statement.

The graph $H$, along with vertex sets $M, \tilde{P}, P, S$, the vertex $o$, and the integer $r$, are obtained by applying the definition of a lunar class. We denote by $T'$ the set of vertices in $H$ to which $T$ is mapped, so that $H[M][T] = H[T'] = G[T]$.

In order to describe our demands $D$, we identify some useful sets of vertices in $H$:

- *Target vertices* are exactly the set $T' = \{t'_1, t'_2, \dots, t'_{|T|}\}$.
- The set of moonlit nodes $\breve{M} = \{\breve{m}_1, \breve{m}_2, \dots, \breve{m}_k\}$ is some arbitrary subset of $\tilde{P}$ with cardinality $k$ exactly.
- The set of sunlit nodes $\breve{S} := N[M] \cup N[P] \setminus (T \cup \breve{M})$.

The intention is that our construction will ensure that, in any configuration $N$ of interest to us, the moonlit nodes $\breve{M}$ (resp. sunlit nodes $\breve{S}$) will be connected by a dynamic link to moon nodes $M$ (resp. sun nodes $S$).

Consider a new graph: the complete bipartite graph with $\breve{S} \cup S$ as vertices and $\breve{S} \times S$ as edges. Let $E_S$ be an arbitrary maximum matching in this bipartite graph. Note that $|S| \geq |\breve{S}|$ by construction, and so $|E_S| = |\breve{S}|$ exactly and each sunlit vertex is incident to exactly one edge in $E_S$. Note that by construction $E_S \cap E(H) = \emptyset$, because none of the sunlit nodes are adjacent to any sun node (by applying a property of lunar graphs). That is, if $(u, v)$ is an edge in $E_S$ then $u$ and $v$ are not adjacent in $H$.

Let $\alpha = (r + \mu + 1)|T|$. Our demand matrix $D$ is fully described by the following:

- $D[u, v] = \alpha$ for each $(u, v) \in E_S$ (sunlight demands),
- $D[o, t] = 1$ for each $t \in T$ (moonlight demands), and
- all other entries of $D$ are zero.

It remains for us to define $\kappa$. As before, we define this as $\kappa_\alpha + \kappa_1$, with $\kappa_\alpha = \mu\alpha|E_S|$ and $\kappa_1 = |T|(r + \mu) + |T| - k$. We note that $\alpha > \kappa_1$. This completes our construction of the instance $(H, \mu, D, \kappa)$.

**Claim 4.19.1.** If $(G, T, k)$ is a yes-instance of Partial Domination then $(H, \mu, D, \kappa)$ is a yes-instance of 1-Switched RRP($\delta = 1$).

*Proof.* An illustration of an optimal configuration is shown in Figure 4.11. Let $X$ be a dominating set of $G[T]$ of cardinality $k$. Denote also by $X' = \{x'_1, x'_2, \dots, x'_k\}$ the corresponding set of vertices in $H$.

Let $N := E_S \cup \{(x'_i, \breve{m}_i) : 1 \leq i \leq k\}$. Clearly, under $N$ each sunlight demand is served at cost exactly $\mu\alpha$ and so all sunlight demands cumulatively are served at cost $|E_S|\mu\alpha = \kappa_\alpha$.

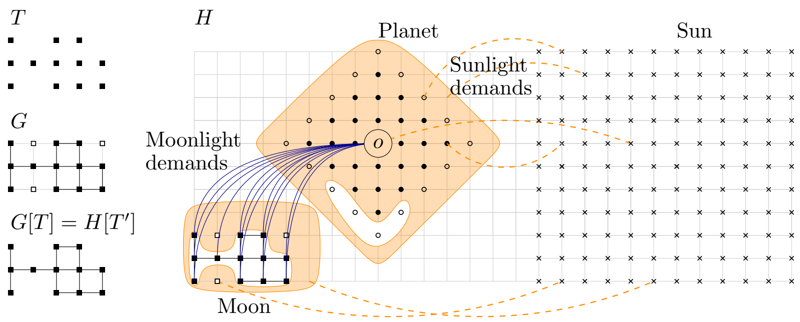


Figure 4.10: Illustration of the demands described in our reduction, for the instance $(G, T, 4)$ of Partial Domination(Grid graphs). Nodes of $T$ (and $T'$ in $H$) are shown as black squares; other nodes of $G$ (and $M$ in $H$) are shown as boxes. Other nodes are shown as in Figure 4.8 earlier. Moonlight demands are shown as solid blue curves. The sunlit nodes $\breve{S}$ are those in orange shaded regions (note exactly 4 nodes of $\tilde{P}$ are not sunlit - these are the moonlit nodes $\breve{M}$). Sunlight demands are drawn as orange arcs (for clarity, only a few are shown).

Further, the $k$ demands from each node $x_i'$ to $o$ are each served at cost exactly $\mu + r$ (by the path $x_i' \dashrightarrow \breve{m}_i \rightsquigarrow_r o$), and the moonlit demand for each of the $|T| - k$ other nodes $t \in T$ is served at cost exactly $1 + \mu + r$ (by the path $t \rightsquigarrow_1 x_i' \dashrightarrow \breve{m}_i \rightsquigarrow_r o$). The claim follows. $\square$

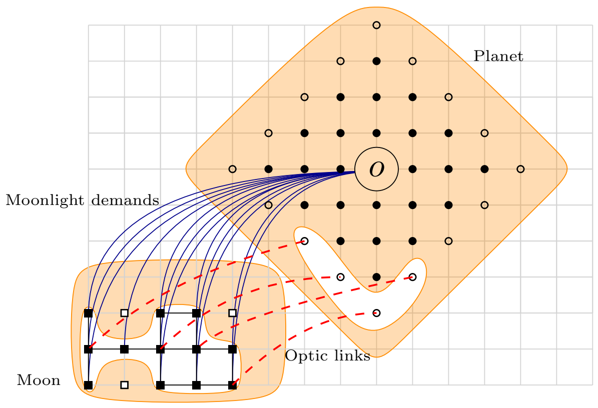


Figure 4.11: Detail of an optimal configuration $N$ for the instance shown in Figure 4.10. Only moonlight demands (solid blue arcs) and dynamic links serving them (dashed red lines) are shown. Note that all moonlit demands are served at cost exactly $\kappa_1$.

Conversely, suppose that $(H, \mu, D, \kappa)$ is a yes-instance of 1-switched RRP ($\delta = 1$) and that $N$ is a configuration and $F$ a set of flow-paths witnessing that the total workload cost is at most $\kappa$.

**Claim 4.19.2.** Each sunlit node has a line of sight (optic link) to a sun node. Formally, every edge $(u, v)$ of $E_S$ is necessarily such that $(u, v) \in N$ and the total workload cost of flow-paths serving the $\alpha$-demands is exactly $\kappa_\alpha$.

*Proof.* Suppose that at least one of the $\alpha$-demands is served via a flow-path at a workload cost of more than $\mu\alpha$; so, it must be at a workload cost of more than $(1+\mu)\alpha$ as we cannot traverse a dynamic link followed immediately by another dynamic link (recall, $\Delta_S = 1$) and by our construction, if $D[u, v] = \alpha$ then there is no static link $(u, v)$. Hence, the total workload cost of flow-paths serving the $\alpha$-demands is at least $(|E_\alpha| - 1)\mu\alpha + (1 + \mu)\alpha = \kappa_\alpha + \alpha$. We have that $\kappa_1 < (r + \mu + 1)|T| = \alpha$. Hence, the total workload cost of all flow-paths of $F$ is strictly greater than $\kappa$ which yields a contradiction and the claim follows. $\square$

**Claim 4.19.3.** Each moonlight demand is served at cost at least $\mu+r$. Furthermore, at most $k$ moonlight demands are served at cost $\mu+r$ exactly.

*Proof.* Consider some moonlight demand; necessarily it has form $D[t,o]=1$ for some $t \in T$.

We first show each moonlight demand is served at cost at least $\mu+r$. Suppose for contradiction that $D[t,o]$ is served at cost strictly less than $\mu+r$. Since $t$ and $o$ are at distance at least $r+3$, the flow-path from $t$ to $o$ must make use of some dynamic link $u \dashrightarrow v$ at cost $\mu$. The static portion of the path therefore must have cost at most $r-1$, which entails that $v \in B(o, r-1) \subsetneq \breve{S}$ (that is, $v$ is a sunlit node). Applying Claim 4.19.2 $v$ is connected by dynamic link to some sun node, so $u \in S$ necessarily. Then the path from $t$ to $u$ has length at least $2r$, yielding the desired contradiction.

Now observe that $D[t,o]$ can be served at cost exactly $\mu+r$ if and only if the dynamic link $t \dashrightarrow \breve{m}$ exists, for some $\breve{m} \in \breve{M}$. Since $|\breve{M}|=k$ exactly by construction, we obtain that this may be the case for at most $k$ nodes in $T$. The claim follows.

$\square$

Since $\kappa_1 = (r+\mu)(k) + (r+\mu+1)(|T|-k)$, we immediately obtain that *exactly* $k$ moonlight demands are served at cost $r+\mu$ and all remaining moonlight demands are served at cost $r+\mu+1$ exactly.

**Claim 4.19.4.** The set $X = \{u : (u,v) \in N \text{ and } v \in \breve{M} \text{ is a moonlit node}\}$ is a dominating set of $H[M]$.

*Proof.* Suppose for contradiction that there is some vertex $t \in T'$ which is neither in $X$ nor adjacent to any vertex in $X$. We show that the cost of serving the demand $D[y,o]=1$ is strictly greater than $r+\mu+2$. Denoting arbitrary nodes $m' \in N[M], p' \in N[P], \tilde{p} \in \tilde{P}$, this flow is routed either:

- Via static links only, along a path of length at least $r+3 > r+\mu+1$: $y \rightsquigarrow_{\geq 1} m' \rightsquigarrow_{\geq 1} p' \rightsquigarrow_1 \tilde{p} \rightsquigarrow_r o$, or
- Via static links and one dynamic link $u \dashrightarrow v$ with:
  - $v$ in $\tilde{P}$, at cost at least $r+\mu+2 > r+\mu+1$: $y \rightsquigarrow_{\geq 2} u \dashrightarrow v \rightsquigarrow_r o$
  - $u$ and $v$ both outside $N[P]$, at cost at least $r+\mu+2 > r+\mu+1$: $y \rightsquigarrow_{\geq 0} u \dashrightarrow v \rightsquigarrow_{\geq 1} p' \rightsquigarrow_1 \tilde{p} \rightsquigarrow_r o$
  - either $u$ or $v$ in $N[P] \setminus \tilde{P}$ (and the other in $S$), at cost at least $2r+\mu$ (recall all vertices in $S$ are distance at least $2r$ from any vertex in $M$).

This contradicts our earlier claim (that all moonlight demands are served at cost at most $r+\mu+1$) and the result follows. $\square$

We note that the construction described takes polynomial time, and the main result follows.

□

Applying our earlier results on lunar graph classes (Lemmas 4.18 and 4.16, Corollary 4.17) together with Theorem 4.13 we obtain that RRP remains hard even when the number of dynamic links admitted on any path is limited to 1.

**Corollary 4.20.** 1-SWITCHED RRP($\delta = 1$)($\mathcal{G}$) is NP-complete if $\mathcal{G}$ is: the class of hypercubes; the class of grid graphs; or the class of toroidal grid graphs.

We emphasize again the value of hypercubes as a prototypical model of interconnection networks, and additionally note that grids and toroidal grids exhibit additional properties which may have been expected to yield a tractable setting, such as planarity and bounded degree. The restriction to $\delta = 1$ in our setting also implies a restriction to $\sigma = 2$ (as 3 alternations along a path would entail a minimum 2 dynamic links along the same) but the converse does not hold; we expect that the case of 1-SWITCHED RRP($\sigma = 2, \delta = 2$) can be shown to be intractable through similar proof techniques, but leave this for future work.

## 4.4 Discussion and Future Work

Taken together, our results comprehensively establish the computational hardness of RRP in practically relevant settings. We establish that the problem remains intractable in several cases where the demand matrix is sparse, the hybrid network is highly structured (in fact node-symmetric) and the weights of links depend only on their medium. Furthermore, in all of our hardness results, the instrument used to "express" NP-completeness is the demand matrix $D$. In the real world, the computational workload for the network is generally expected to vary significantly with time, unlike the network's hardware, which (in addition to its structural properties already discussed) does not rapidly change. Our results are in this sense closely relevant to the hardness of the real world reconfigurable routing problem.

We take this opportunity to identify some specific questions we have left open, as well as several more general avenues for future work in this area. First, it would be interesting to study the restriction of the problem to cases where $\Delta_S$ is greater than 1 and $\mu$ is a fixed constant. Results in this setting would "bridge the gap" between Theorems 4.1 and 4.2, and Theorems 4.6 and 4.19. Analogously, there is a gap for 1-SWITCHED RRP on hypercubes between $\sigma = 0$ (which is solvable in polynomial time) and $\sigma = 3$ (which is an intractable case). The complexity of the problem with $\sigma = 1$ and $\sigma = 2$ remains open for hypercubes (note that results for arbitrary networks do exist when $\sigma = 2$, as shown in Table 4.1). Our Theorem 4.19 in some sense sits between these two open cases, since the restriction to $\sigma = 1$ entails a restriction to $\delta = 1$, which itself entails a restriction to $\sigma = 2$.

Secondly, the present work considers only exact computation. In [36] the authors establish inapproximability within $\Omega(\log n)$ for RRP in a more permissive setting (making use of variable link weights). However, the empty solution (there are no dynamic links and all demands are routed through the static network only) is a $\frac{\log n}{\mu}$-approximation for $\Delta_S$-Switched RRP on hypercubes. (This follows straightforwardly from hypercubes having logarithmic diameter.) It would be interesting to see what (in)approximability results can be derived in our model with fixed link weights, with and without restrictions to realistic topologies.

Lastly, parameterized algorithms may provide more fine-grained insights into the computational complexity of reconfigurable routing. Our Theorems 4.1 and 4.2 establish that structural parameters of the static network, such as treewidth, are insufficient to yield fixed-parameter tractable (fpt) algorithms (unless P=NP). However, it would be interesting to see whether it is possible to obtain an fpt algorithm by additionally parameterizing by the sum of the demand matrix $D$; some structural parameters for the digraph representation of the demands, $D'$; the dynamic link weight $\mu$; or a combination of these.

# Chapter 5

# Detours and Distractions.

This chapter gathers four works which I contributed to during my PhD which, for different reasons, I wanted to include in the thesis but felt did not warrant a full chapter each.

First, in chapter 5a, I present my contributions to research around Boolean Networks which I undertook together with Maximilien Gadouleau. This is the only subchapter of chapter 5 based on work which has been published.

Second, in chapter 5b, I present the result of a joint project with Anouk Sommer, whom I hosted for a summer research internship in 2024.

Third, in chapter 5c, I present some preliminary findings which extend some notions developed while working on the research in chapter 4, but which are sufficiently distant from the original setting as to merit their own section.

And fourth, in chapter 5d, I give a proof that a specific Constraint Satisfaction Problem (CSP) is NP-complete – which also was originally devised to prove a result for chapter 4 but was ultimately unnecessary for that purpose.

# Chapter 5a

# Maximal Independent Sets and Boolean Networks

*This chapter is based on joint work [50] with Maximilien Gadouleau, an associate professor in the Computer Science department at Durham. In order to constrain this subchapter to 23 pages (a pleasant number [139]) while still providing the flavor of the investigation it is based on, a number of results are presented here without proofs (in which case the reader is referred to the full published paper [50]), and an extension to digraphs (Section 7 in [50]) is omitted. Prioritized for inclusion are those proofs which the author was most closely involved with, and those which are most consistent with the theme of the thesis: exploring the complexity of eventful graph problems.*

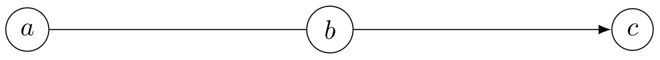
## 5a.1 Introduction

**The MIS algorithm** A simple greedy algorithm to find a maximal independent set (MIS) in a graph starts with the empty set and visits every vertex, adding it to the set if and only if none of its neighbours are already in the set. We shall refer to it as the MIS algorithm. Because the MIS algorithm always terminates in a maximal independent set, it has been the subject of a stream of work (see [140] and references therein).

A core feature of the classical MIS algorithm is that the starting set of vertices is the empty set. However, the seminal observation of this chapter is that this constraint can be lifted. Indeed, starting from any set of vertices and visiting each vertex once, removing a vertex if one of its neighbours already appears in the set, one always terminates at an independent set. Moreover, starting from any independent set and visiting each vertex once, one always terminates at a MIS. Thus, iterating over the vertex set twice is sufficient to obtain a MIS from any starting set of vertices.

As such, the scope of this chapter is the generalisation of the MIS algorithm, where one can start with any (not necessarily independent) set of vertices, and one can visit vertices in any order with possible repetitions. Of course, some sequences of vertices will guarantee reaching a MIS from any starting configuration – we shall call those fixing words, while others will not, as in Example 5a.1 below.

**Example 5a.1.** Consider the path on three vertices:



This graph has two MIS, namely $\{a, c\}$ and $\{b\}$. Starting at the empty set, the MIS algorithm terminates at $\{b\}$ if the sequence begins with $b$ (i.e. $w = bac$ or $w = bca$) or at $\{a, c\}$ otherwise (i.e. $w \in \{abc, acb, cab, cba\}$).

For this graph, *abc* is not a fixing word: if one starts from the set $\{b, c\}$, then one terminates at $\{c\}$. However, *acb* is a fixing word: if the starting set contains $b$, then one terminates at $\{b\}$; otherwise one terminates at $\{a, c\}$.

Finally, for this graph, the words $w = abcabc$ and $w = acbacb$ are fixing words, as are any words of the form $w = w^1 w^2$, where $w^1$ and $w^2$ are permutations of the vertex set.

**Contributions for graphs** Below we give a summary of our contributions for the MIS algorithm on graphs.

When starting from the empty set, the MIS algorithm is able to reach any possible maximal independent set (if the algorithm goes through the MIS first, then it would terminate with that MIS). However, the empty set is not the only set with that property: the full set of vertices also allows that (this time, if the algorithm finishes with a MIS). In Theorem 5a.7, we prove that deciding whether a set of vertices can reach every MIS is **coNP**-complete.

As we showed in Example 5a.1, though iterating over the whole set of vertices twice is always sufficient to reach a MIS, it is not always necessary. Consequently, we ask: what are the sequences of vertices which always reach a MIS, regardless of the starting set of vertices? In Theorem 5a.11, we prove that deciding whether a sequence offers that guarantee is **coNP**-complete.

Since the MIS algorithm visits each vertex exactly once, we also consider permutations of vertices that are guaranteed to reach a MIS: *permises*. In Theorem 5a.20, we prove that deciding whether a permutation of vertices is a permis is **coNP**-complete. A graph that admits a permis is called permissible. Not all graphs are permissible; the smallest non-permissible graph is the heptagon. We exhibit large classes of permissible and non-permissible graphs. In particular, we introduce near-comparability graphs and classify them in Theorem 5a.23; they naturally generalise comparability graphs and can be recognised in polynomial time. We prove that near-comparability graphs are permissible in Proposition 5a.22. In Theorem 5a.34, we prove that deciding whether a graph is permissible is **coNP**-hard. There is no obvious candidate for a no-certificate, so it may be that the problem is not actually in **coNP**.

In some situations, one can skip some vertices and still guarantee a MIS is reached. For instance, in the complete graph, one can simply update all but one vertex and still reach a maximal independent set, from any starting configuration. We prove in Theorem 5a.18 that deciding whether a given set of vertices can be skipped is **coNP**-complete. We also prove in Theorem 5a.19 that deciding whether *any* vertices can be skipped is **coNP**-complete.

**Boolean networks** Our main tool is Boolean networks. A configuration on a graph $G = (V, E)$ is $x \in \{0,1\}^V$, i.e. the assignment of a Boolean state to every vertex of the graph. A Boolean network is a mapping $\mathsf{F} : \{0,1\}^V \to \{0,1\}^V$ that acts on the set of configurations. Boolean networks are used to model networks of interacting entities. As such, it is natural to consider a scenario wherein the different entities update their state at different times. This gives rise to the notion of sequential (or asynchronous) updates, by updating the state of one vertex at a time; a word $w$ then gives the order in which those vertices are updated (with repeats allowed in general). Since the original works by Kauffman [141] and Thomas [142], asynchronous updates have been widely studied, both in terms of modelling purposes and of dynamical analysis (see [143, 144] and references therein). The problem of whether a Boolean network converges (sequentially) goes back to the seminal result by Robert on acyclic interaction graphs [145]; further results include [146, 147, 148]. Recently, [149] introduced the concept of a fixing word: a word $w$ such that updating vertices according to $w$ will always lead to a fixed point, regardless of the initial configuration. Fixing words are a natural feature of Boolean networks, for two main reasons. Firstly, almost all networks with a fixed point, and hence a positive asymptotic proportion of all networks, have fixing words [150]. Secondly, large classes of Boolean networks, including monotone networks and networks with acyclic interaction graphs, have short fixing words (of length at most cubic in $|V|$) [149, 151].

We refer to the Boolean network where the update function is the conjunction of all the negated variables in the neighbourhood of a vertex as the MIS network on the graph, i.e. $\mathsf{M} : \{0,1\}^V \to \{0,1\}^V$ with $\mathsf{M}(x)_v = \bigwedge_{u \sim v} \neg x_u$ for all $v \in V$. The MIS network was highlighted in [152, 153], where the fixed points of different conjunctive networks on (directed) graphs are studied. In particular, [152] shows that the set of fixed points of the MIS network is the set of (configurations whose supports are) maximal independent sets of the graph. It is further shown in [153] that for square-free graphs, the MIS network is the conjunctive network that maximises the number of fixed points.

The MIS algorithm can be interpreted in terms of Boolean networks as follows: starting with the all-zero configuration $x$, update one vertex $v$ at a time according to the update function $\mathsf{M}(x)_v = \bigwedge_{u \sim v} \neg x_u$. Once all vertices have been updated, we obtain the final configuration $y$ where the set of ones is a MIS, regardless of the order in which the vertices have been updated. As such, fixing words of the MIS network correspond to sequences of vertices that guarantee that the MIS algorithm terminates for any starting set of vertices. The seminal observation of this chapter is that for any permutation $w$, the word $ww$ is a short fixing word (of length $2|V|$).

**Self-stabilization and distributed computing** Some of our results may be applied in the context of distributed computing. Similar to Boolean networks, distributed algorithms produce an output through local, independent updates of nodes in a fixed topology. Asynchronous models for distributed computing do not assume a bound on message delay [41], making them less relevant here. Consequently, we focus on synchronous models, in which time is discrete and all nodes perform a SEND-RECEIVE-UPDATE loop synchronously at each time step. The algorithm executed at all nodes is identical, and the state of some node at time $t$ depends only on the state of (all nodes in) its inclusive neighbourhood at time $t-1$. Some key differences with the Boolean Network setting are worth emphasizing. For example, in standard models for synchronous distributed computation: nodes each have an (unbounded-size) internal state, and may solve arbitrarily hard problems during their UPDATE; messages sent may differ from the sender's state; nodes may choose not to SEND anything at all; and updates occur synchronously.

The problem of finding a MIS has been a focus of much study in this setting, including in the LOCAL [43], CONGEST [154] and Beeping models [155, 42]. LOCAL is characterized by its unrestricted message size, whereas CONGEST limits messages to $O(\log |V|)$ bits per outgoing edge. The Beeping model is a significant restriction, in which nodes can communicate only via beeps (which are indistinguishable) and silence [156]. We refer the interested reader to [157] for a more complete treatment of this model's variants, which also includes a discussion of the distributed MIS problem in Sections 4.5 and 6.2.

An algorithm or procedure is said to be "self-stabilizing" if it is guaranteed to reach a legitimate state regardless of its initial state, and additionally will never reach an illegitimate state from a legitimate state [158, 159]. This notion has been integrated into the design of distributed algorithms [160] and is explicitly identified as a feature of the Beeping MIS algorithm given in [155].

Our results do not directly apply to these models; in particular, we assume (and sometimes exploit) asynchronous and instantaneous updates. That is, each vertex's local update is immediately "visible" to all its neighbours. This differs from standard models of distributed computing, which generally incorporate some transmission delay (which is typically one unit in synchronous models, and controlled by an adversary in asynchronous models).

To emulate the MIS network $\mathsf{M}$ studied in the present work, it would then be sufficient for a distributed model to support: asynchronous updates (scheduled by an adversary or a helper) and instantaneous transmission. We call the minimum length of time within which each node updates at least once a *phase*. In the adversarial setting, our seminal observation translates to the fact that this protocol necessarily reaches a MIS within two phases from any starting configuration and self-stabilizes. In the helpful setting, a Permis is an update schedule which guarantees self-stabilization within a single phase. By Theorem 5a.11, it is **coNP**-complete to determine whether some update schedule satisfies this property; by Theorem 5a.20, the problem remains **coNP**-complete even if the schedule is guaranteed to contain every node exactly once; and by Theorem 5a.34 it is **coNP**-hard to determine whether any such update schedule exists at all for the given network. If the helpful scheduler is limited to some subset of nodes, Theorem 5a.18 means it is **coNP**-hard to determine whether there is a self-stabilizing schedule which uses only that subset of nodes. Furthermore, Theorem 5a.19 entails that deciding whether there exists any such schedule using $n-1$ nodes (even allowing repetitions) is **coNP**-hard.

**Constituencies** The main tool for the hardness results in this chapter is that of a constituency. A constituency is a set of vertices of a graph that is dominated by an independent set, i.e. $S$ is a constituency if there exists an independent set $I$ such that $S \subseteq N(I)$. We believe that the constituency problem is of independent interest for a couple of reasons. Firstly, this is a natural definition for a set of vertices, but to the best of the authors' knowledge, it has not been considered in the literature yet. Secondly, the CONSTITUENCY problem asks whether a set $S$ is a constituency. Unlike problems like CLIQUE, INDEPENDENT SET or VERTEX COVER, the CONSTITUENCY problem does not rely on an integer parameter. Nonetheless, CONSTITUENCY is **NP**-complete, while the problem of deciding whether a set is a clique (or independent set, or vertex cover) is clearly in **P**. As such, CONSTITUENCY provides a natural intractable graph problem whose input does not include an integer. We heavily use CONSTITUENCY and its variants in our hardness proofs, and we believe that this problem could be used more broadly for reductions in the wider graph theory community.

We take this opportunity to mention a related class of problems – so-called *extension* problems, in which a designated set of elements must be included (resp. excluded) in the solution (for example, in a maximal independent set, a minimal dominating set, or a maximal matching), the size of which is not specified. We refer the interested reader to [161] and the references therein for an overview.

**Outline** The rest of the chapter is organised as follows. Some necessary background is given in Section 5a.2. Constituencies and districts are introduced in Section 5a.3, where some decision problems based on those are proved to be **NP-** or **coNP-**complete. The configurations that allow to reach any possible maximal independent set are determined in Section 5a.4. Fixing words, fixing sets, and permises for the MIS network are studied in Section 5a.5. Classes of permissible and non-permissible graphs are given in Section 5a.6. Finally, some conclusions and possible avenues for future work are given in Section 5a.7.

## 5a.2 Preliminaries

### 5a.2.1 Graphs and digraphs

Most of our contributions will focus on (undirected) graphs. However, when we also make use of directed graphs in some arguments (particularly pertaining to orientations of undirected graphs), so we shall view graphs as natural special cases of digraphs. As such, we give the background on graphs and digraphs in its full generality, i.e. for digraphs first, and then we make some notes about the special case of graphs.

By digraph, we mean an irreflexive directed graph, i.e. $G = (V, E)$ where $E \subseteq V^2 \setminus \{(v, v) : v \in V\}$. We use the notation $u \to v$ to mean that $(u, v) \in E$. We say an edge

$(u, v) \in E$ is **symmetric** if $(v, u)$ is also an edge, and **oriented** otherwise. We will sometimes emphasize that $(u, v)$ is symmetric by writing it $uv$ instead. For a vertex $v$, the open in-neighbourhood, closed in-neighbourhood, open out-neighbourhood and closed out-neighbourhood of the vertex $v$ are respectively defined as

$$N^{\text{in}}(v) = \{u \in V : u \to v\}, \quad N^{\text{in}}[v] = N^{\text{in}}(v) \cup \{v\},$$
$$N^{\text{out}}(v) = \{u \in V : v \to u\}, \quad N^{\text{out}}[v] = N^{\text{out}}(v) \cup \{v\}.$$

All of those are generalised to sets of vertices, e.g. $N^{\text{in}}(S) = \bigcup_{s \in S} N^{\text{in}}(s)$. Clearly, all notations above should reflect the dependence on the digraph $G$, e.g. $N^{\text{in}}(v; G)$; we shall omit that dependence on any notation when the digraph is clear from the context.

For a digraph $G = (V, E)$ and set of vertices $S \subseteq V$, we call the digraph $(S, \{(u, v) : (u, v) \in E \wedge \{u, v\} \subseteq S\})$ the **induced subgraph** on $S$, which we denote $G[S]$. We denote $G - S$ the digraph $G[V \setminus S]$. A **path** is a sequence of edges $v_1 \to v_2 \to \cdots \to v_k$ where all vertices are distinct; a **cycle** in a digraph is a sequence of edges $v_1 \to v_2 \to \cdots \to v_k \to v_1$ where only the first and the last vertices are equal. A digraph is **strong** if for all vertices $u$ and $v$, there is a path from $u$ to $v$. A **strong component** of $G$ is a subset of vertices $S$ such that $G[S]$ is strong, but $G[T]$ is not strong for all $T \supsetneq S$. A digraph is **acyclic** if it has no cycles. An acyclic digraph has a **topological order**, whereby $u \to v$ only if $u \leq v$. For instance, the digraph where each vertex is a strong component of $G$ and $C \to C'$ if and only if $u \to u'$ for some $u \in C$, $u' \in C'$ is acyclic. If $C \to C'$ in that digraph, we say that $C$ is a **parent component** of $C$; a strong component without any parent is called an **initial component**.

We say that two vertices $u$ and $v$ are **closed twins** if $N^{\text{in}}[u] = N^{\text{in}}[v]$. Accordingly, we say that the vertex $m$ is a **benjamin** of $G$ if there is no vertex $v$ with $N^{\text{in}}[v] \subset N^{\text{in}}[m]$. We denote the set of benjamins of $G$ by $\mathrm{B}(G)$ and the corresponding induced subgraph by $G_{\mathrm{B}} = G[\mathrm{B}(G)]$. We say that a set of vertices $S$ is **tethered** if there is an edge $st$ between any $s \in S$ and any $t \in T = N(S) \setminus S$.

A digraph is **undirected** if all its edges are symmetric; which we shall simply call a **graph**. We then denote $N(v) = N^{\text{in}}(v) = N^{\text{out}}(v)$, which we call the **neighbourhood** of $v$. A strong graph is called **connected**, and the (initial) strong components of a graph are called its **connected components**. In a graph, if $u \to v$, then $v \to u$, which we shall denote by $u \sim v$.

### 5a.2.2 Boolean networks

A **configuration** on a digraph $G = (V, E)$ is $x \in \{0, 1\}^V = (x_v : v \in V)$, where $x_v \in \{0, 1\}$ is the state of the vertex $v$ for all $v$. We denote $\mathbf{1}(x) = \{v \in V : x_v = 1\}$ and $\mathbf{0}(x) = \{v \in V : x_v = 0\}$. Conversely, for any set of vertices $S \subseteq V$, the **characteristic vector** of $S$ is the configuration $x$ such that $\mathbf{1}(x) = S$. For any set of vertices $S \subseteq V$, we denote $x_S = (x_v : v \in S)$. We denote the all-zero (all-one, respectively) configuration by 0 (by 1, respectively), regardless of its length.

We consider the following kinds of sets of vertices of, and accordingly configurations on, a digraph $G$:

1. An **independent set** $I$ is a set such that $(i,j) \notin E$ for all $i,j \in I$. (In other words, $N^{\text{out}}(I) \cap I = \emptyset$.) Every digraph $G$ has an independent set, namely the empty set $\emptyset$. The collection of characteristic vectors of independent sets of $G$ is denoted by $\mathrm{I}(G)$.

2. A **dominating set** $D$ is a set such that for every vertex $v \in V$, either $v \in D$ or there exists $u \in D$ such that $(u,v) \in E$. (In other words, $N^{\text{out}}(D) \cup D = V$.) Every digraph $G$ has an dominating set, namely $V$. The collection of characteristic vectors of dominating sets of $G$ is denoted by $\mathrm{D}(G)$.

3. A **kernel** $K$ is a dominating independent set. (In other words, $N^{\text{out}}(K) \uplus K = V$.) Not all digraphs have a kernel, for instance the directed cycle $\vec{C}_n$ (with vertex set $\mathbb{Z}_n$ and edges $(v, v{+}1)$ for all $v \in \mathbb{Z}_n$) does not have a kernel whenever $n \geq 3$ is odd. The collection of characteristic vectors of kernels of $G$ is denoted by $\mathrm{K}(G)$.

4. If $G$ is a graph, then a kernel is a **maximal independent set** of $G$, i.e. an independent set $I$ such that there is no independent set $J \supset I$. Every graph has a maximal independent set. In order to highlight this special case of particular importance to this chapter, the collection of characteristic vectors of maximal independent sets of $G$ is denoted by $\mathrm{M}(G)$.

Let $w = w_1 \dots w_l \in V^*$ be a sequence of vertices, or briefly a **word**. For any $a, b \in \{1, \dots, l\}$, we denote $w_{a:b} = w_a \dots w_b$ if $a \leq b$ and $w_{a:b}$ is the empty sequence if $a > b$. We also denote by $[w] = \{u \in V : \exists j \ w_j = u\}$ the set of vertices that $w$ visits. For any $S \subseteq V$, the subword of $w$ induced by $S$, denoted by $w[S]$, is obtained by deleting all the entries in $w$ that do not belong to $S$; alternatively, it is the longest subword of $w$ such that $[w[S]] \subseteq S$. A **permutation** of $V$ is a word $w = w_{1:n}$ such that $[w] = V$ and $w_a \neq w_b$ for all $a \neq b$.

A **Boolean network** is a mapping $\mathsf{F} : \{0,1\}^V \to \{0,1\}^V$. For any Boolean network $\mathsf{F}$ and any $v \in V$, the update of the state of vertex $v$ is represented by the network $\mathsf{F}^v : \{0,1\}^V \to \{0,1\}^V$ where $\mathsf{F}^v(x)_v = \mathsf{F}(x)_v$ and $\mathsf{F}^v(x)_u = x_u$ for all other vertices $u$. We extend this notation to words as follows: if $w = w_1 \dots w_l$ then

$$\mathsf{F}^w = \mathsf{F}^{w_l} \circ \dots \circ \mathsf{F}^{w_2} \circ \mathsf{F}^{w_1}.$$

Unless otherwise specified, we let $x$ be the initial configuration, $w = w_1 \dots w_l$ be a word, $y = \mathsf{F}^w(x)$ be the final configuration, and for all $0 \leq a \leq l$, $y^a = \mathsf{F}^{w_{1:a}}(x)$ be an intermediate configuration, so that $x = y^0$ and $y = y^l$.

If there is a word $w$ such that $y = \mathsf{F}^w(x)$, we say that $y$ is **reachable** from $x$, and we write $x \mapsto_{\mathsf{F}} y$. For any two configurations $x$ and $y$, we denote $\Delta(x,y) = \{v \in V : x_v \neq y_v\}$. An **$\mathsf{F}$-geodesic** from $x$ to $y$ is a word $w$ such that $y = \mathsf{F}^w(x)$, $[w] = \Delta(x,y)$ and $w_a \neq w_b$ for all $a \neq b$, i.e. $w$ visits every vertex $v$ where $x$ and $y$ differ exactly once, and does not visit any other vertex. If there exists a geodesic from $x$ to $y$, we denote it by $x \xmapsto{\text{geo}}_{\mathsf{F}} y$.

The set of **fixed points** of $\mathsf{F}$ is $\mathrm{Fix}(\mathsf{F}) = \{x \in \{0,1\}^V : \mathsf{F}(x) = x\}$. It is clear that $x \in \mathrm{Fix}(\mathsf{F})$ if and only if $\mathsf{F}^w(x) = x$ for any word $w$, i.e. a "parallel" fixed point is also a "sequential" fixed point. The word $w$ is a **fixing word** for $\mathsf{F}$ [149] (and we say that $w$ **fixes** $\mathsf{F}$) if for all $x$, $\mathsf{F}^w(x) \in \mathrm{Fix}(\mathsf{F})$ (see [149] for some examples of fixing words). A Boolean network is **fixable** if it has a fixing word.

## 5a.3 Constituencies and districts

In this section, we introduce two kinds of sets of vertices, namely constituencies and districts, and we determine the complexity of some decision problems related to them. Even though both concepts will be useful to the sequel of this chapter (an intuition behind the role of constituencies is given in the introduction of Section 5a.5), we believe that the concept of constituency in particular is a natural property and is interesting to the wider graph theory community.

### 5a.3.1 Constituencies

Let $G = (V, E)$ be a graph. A subset $S$ of $V$ is a **constituency** of $G$ if there exists an independent set $I$ such that $S \subseteq N(I)$ (note that this requires that $S \cap I = \emptyset$). The following are equivalent for a set of vertices $S \subseteq V$ (the proof is easy and hence omitted):

1. $S$ is a constituency of $G$, i.e. there exists an independent set $I$ of $G$ such that $S \subseteq N(I)$;
2. $V \setminus S$ contains a maximal independent set of $G$;
3. there exists a maximal independent set $M$ of $G$ such that $M \cap S = \emptyset$;

A **non-constituency** is a set of vertices that is not a constituency. The Constituency (Non-Constituency, respectively) problem asks, given a graph $G$ and set $S$, whether $S$ a constituency (a non-constituency, respectively) of $G$.

(Non-)Constituency

**Input:** A graph $G = (V, E)$ and a set of vertices $S \subseteq V$.

**Question:** Is $S$ a (non-)constituency of $G$?

**Theorem 5a.2.** Constituency is **NP**-complete.

*Proof.* Membership of **NP** is known: the yes-certificate is an independent set $I$ such that $S \subseteq N(I)$.

The hardness proof is by reduction from Set Cover, which is **NP**-complete [10]. In Set Cover, the input is a finite set of elements $X = \{x_1, \ldots, x_n\}$, a collection

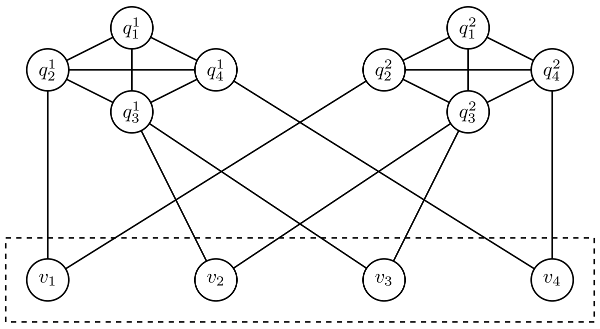


Figure 5a.1: Illustration of the reduction from SET COVER to CONSTITUENCY (the set $S$ is the vertices in the dashed box). Here the SET COVER instance has $C_1 = \emptyset, C_2 = \{x_1\}, C_3 = \{x_2, x_3\}, C_4 = \{x_4\}$, with $k = 2$. Observe that both the SET COVER instance and the CONSTITUENCY instance are no-instances.

$C = \{C_1, C_2, \ldots, C_m\}$ of subsets of $X$, and an integer $k$. The question is whether there exists a subset $Y \subseteq C$ of cardinality at most $k$ such that $\cup_{C_i \in Y} C_i = X$.

We first construct the graph $G$ on $n + mk$ vertices. $G$ consists of: vertices $Q_j = \{q_j^1, \ldots, q_j^k\}$, for each $j \in [m]$; vertices $v_i$ for each $i \in [n]$; edges from each vertex in $Q_j$ to $v_i$, whenever $x_i \in C_j$; edges connecting $\{q_1^l, q_2^l, \ldots, q_m^l\}$ in a clique, for each $l \in [k]$. Let the target set $S = \{v_1, \ldots, v_n\}$. This concludes our construction; an illustrative example is shown in Figure 5a.1.

We now show that if $(X, C, k)$ is a yes-instance of SET COVER, then $(G, S)$ is a yes-instance of CONSTITUENCY. Let $Y \subseteq C$ be a set cover of $X$ of cardinality at most $k$. We obtain the set $I$ as follows:

$$I = \{q_j^a : C_j \text{ is the } a\text{th element of } Y\}.$$

Note that every vertex in $I$ exists in $G$ since $Y$ has cardinality at most $k$ (if $|Y| = k$ then the last subset to appear in $Y$ is its $k$th element exactly). Further, $I$ is an independent set, since by construction every vertex $q_j^a$ is adjacent to some other vertex $q_l^b$ if and only if $a = b$. Lastly, every vertex $v_i \in Y$ is incident to some vertex in $I$; for any $i$, $\exists j : v_i \in C_j$. Then necessarily $\exists a : q_j^a \in I$, and by construction $(v_i, q_j^a)$ is an edge in $G$.

Conversely, if $(G, S)$ is a yes-instance of CONSTITUENCY then $(X, C, k)$ is a yes-instance of SET COVER. Let $I$ be an independent set in $G$ which dominates $S$. By construction of $G$, $I$ has cardinality at most $k$. Suppose otherwise, for contradiction - then by the pigeon-hole principle there is some clique $C_j$ such that $|C_j \cap I| \geq 2$, contradicting that $I$ is an independent set. We obtain the set $Y$ of cardinality $|I|$ as follows:

$$Y = \{C_j : \exists a \text{ such that } q_j^a \in I\}.$$

We now show $Y$ is a set cover of $X$. For each $i \in [n]$, $v_i$ must be adjacent to some vertex in $I$; denote this vertex $q_j^a$ - now by construction $x_i$ is in the set $C_j$, and $C_j \in Y$.

□

**Corollary 5a.3.** NON-CONSTITUENCY is **coNP**-complete.

We make four remarks about constituencies. Let $G$ be a graph, $S$ be a subset of its vertices, and $T = N(S) \setminus S$. First, if $G - S$ has an isolated vertex $t$, then $S$ is a constituency of $G$ if and only if $S \setminus N(t)$ is a constituency of $G - t$. Second, whether $S$ is a constituency of $G$ is independent of the edges in $G[S]$. As such, we can (and shall) reduce ourselves to either of two canonical types of instances $(G, S)$ of Constituency (and of course, of Non-Constituency as well):

**Complete type:** $G[S]$ is complete and $G - S$ has no isolated vertices.

**Empty type:** $G[S]$ is empty and $G - S$ has no isolated vertices.

Third, $S$ is a constituency of $G$ if and only if $S$ is a constituency of $G[S \cup T]$. Therefore, we could reduce ourselves to the case where $V = S \cup N(S)$; however, this assumption shall be unnecessary in our subsequent proofs and as such we shall not use it. Fourth, if $S$ is a constituency of $G$ then every subset of $S$ is also a constituency of $G$.

## 5a.3.2 Districts

A subset $T$ of vertices of a graph $G$ is a **district** of $G$ if there exists $v \in V \setminus T$ such that $T \cap N(v)$ is a constituency of $G - v$. A **non-district** is a set of vertices that is not a district. The District (Non-District, respectively) decision problem asks, given a graph $G$ and a set $T$, whether $T$ is a district (a non-district, respectively) of $G$.

(Non-)District

**Input:** A graph $G = (V, E)$ and a set of vertices $T \subseteq V$.

**Question:** Is $T$ a (non-)district of $G$?

**Theorem 5a.4.** District is **NP**-complete.

*Proof.* Membership of **NP** is known: the yes-certificate is a vertex $v$ and a set of vertices $I$ such that $v \notin I \cup T$, $I$ is an independent set, and $T \cap N(v) \subseteq N(I)$.

The hardness proof is by reduction from Constituency, which is **NP**-complete, as proved in Theorem 5a.2. Let $(G, S)$ be an instance of Constituency, and construct the instance $(\hat{G}, \hat{S})$ of District as follows.

Let $G = (V, E)$ and denote $T = V \setminus S$. Then consider a copy $T' = \{t' : t \in T\}$ of $T$ and an additional vertex $\hat{v} \notin V \cup T'$. Let $\hat{G} = (\hat{V}, \hat{E})$ with $\hat{V} = V \cup T' \cup \{\hat{v}\}$ and $\hat{E} = E \cup \{tt' : t \in T\} \cup \{s\hat{v} : s \in S\}$, and $\hat{S} = S \cup T'$. This construction is illustrated in Figure 5a.2.

We only need to prove that $S$ is a constituency of $G$ if and only if $\hat{S}$ is a district of $\hat{G}$. Firstly, if $S$ is a constituency of $G$, then there exists an independent set $I$ of $G$ such that $S \subseteq N(I; G)$. Then $\hat{S} \cap N(\hat{v}; \hat{G}) = S$ is contained in $N(I; \hat{G} - \hat{v})$, thus $\hat{S}$ is a district of $\hat{G}$.

Conversely, if $\hat{S}$ is a district of $\hat{G}$, then there exists $u \in \hat{V} \setminus \hat{S}$ such that $\hat{S} \cap N(u; \hat{G})$ is a constituency of $\hat{G} - u$. Then either $u = \hat{v}$ or $u \in T$. Suppose $u = t \in T$, then

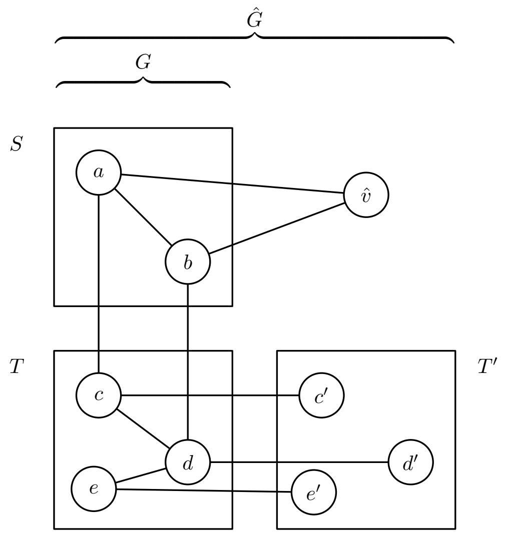


Figure 5a.2: Example reduction from a no-instance of Constituency $(G, S)$ to the corresponding no-instance of District $(\hat{G}, \hat{S})$, with $\hat{S} := S \cup T'$.

$t' \in \hat{S}$ is an isolated vertex of $G - t$, hence $\hat{S} \cap N(t; \hat{G})$ is not a constituency of $\hat{G} - t$. Therefore, $u = \hat{v}$ and there exists an independent set $\hat{I}$ of $\hat{G} - \hat{v}$ such that $\hat{S} \cap N(\hat{v}; \hat{G}) = S$ is contained in $N(\hat{I}; \hat{G})$. Since $S \subseteq V$ and $N(S; \hat{G} - \hat{v}) \subseteq V$, we obtain $S \subseteq N(\hat{I} \cap V; \hat{G} - \hat{v}) \cap V = N(\hat{I} \cap V; G)$, where $I = \hat{I} \cap V$ is an independent set of $G$. Thus, $S$ is a constituency of $G$.

□

**Corollary 5a.5.** Non-District is **coNP**-complete.

If $T$ is a district of $G$, then any subset of $T$ is also a district of $G$. Therefore, any superset of a non-district is also a non-district. Furthermore, every graph $G$ has a trivial non-district, namely $V$. The Non-Trivial Non-District problem asks whether $G$ has any other non-district. We provide some illustrative instances in Figure 5a.3. We need only consider sets $W$ with $|W| = n - 1$. For $C_4$ and $C_3$, we can by symmetry assume $W = V \setminus \{a\}$, and then for $C_4$ $\{c\}$ is an independent set which dominates $N(a) \cap W$, whereas for the $C_3$ there are no vertices outside $N(a) \cap W$ and hence $\{b, c\}$ is a non-district. Similarly, for $P_3$, $W = V \setminus \{b\}$ is a non-trivial non-district.

Non-Trivial Non-District

**Input:** A graph $G = (V, E)$.

**Question:** Does there exist a non-district $S \neq V$ of $G$?

**Theorem 5a.6.** Non-Trivial Non-District is **coNP**-complete.

*Proof.* Since any superset of a non-district is also a non-district, $G$ has a non-district $S \neq V$ if and only if there exists $v \in V$ such that $V \setminus \{v\}$ is a non-district of $G$. Therefore, Non-Trivial Non-District is in **coNP**, where the no-certificate is a collection $(I_v : v \in V)$ such that $I_v$ is an independent set of $G - v$ and $N(v) \subseteq N(I_v)$ for all $v$.

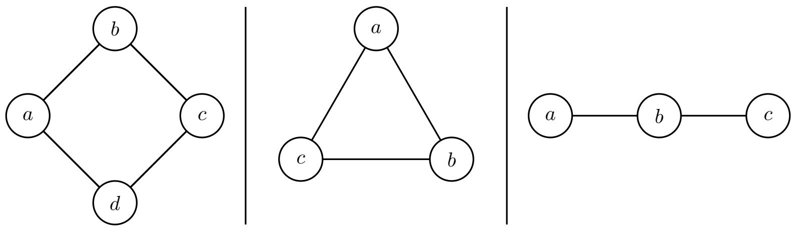


Figure 5a.3: Some example instances of the Non-Trivial Non-District problem. $C_4$ (left) is a no-instance, whereas $C_3$ (center) and $P_3$ (right) are yes-instances.

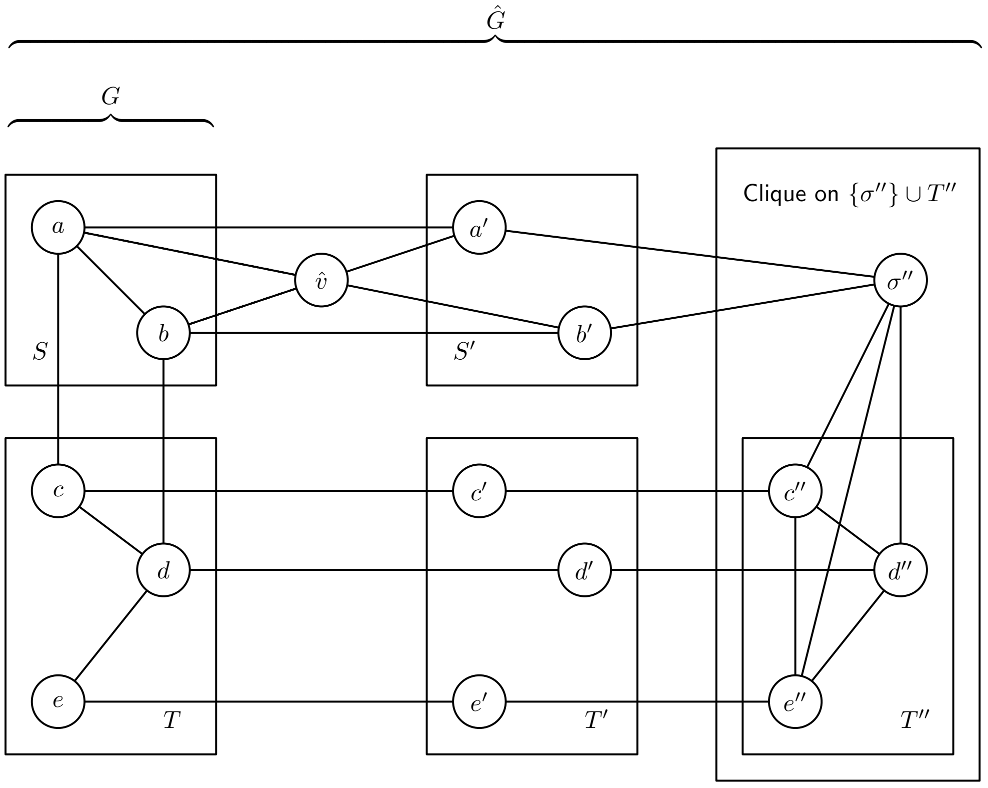


Figure 5a.4: Illustration of the reduction from Non-Constituency to Non-Trivial Non-District.

.

The hardness proof is by reduction from Non-Constituency, which is **coNP**-complete by Corollary 5a.3. Let $(G = (V, E), S)$ be an instance of Non-Constituency of complete type (i.e. where $S$ is a clique in $G$) and denote $T = V \setminus S$. Let $V' = \{v' : v \in V\}$ be a copy of $V$, $T'' = \{t'' : t \in T\}$ be a second copy of $T$, and $\sigma''$ and $\hat{v}$ be two additional vertices. For any $A \subseteq V$, we denote $A' = \{a' : a \in A\}$. Let $\hat{G} = (\hat{V}, \hat{E})$ with $\hat{V} = V \cup V' \cup T'' \cup \{\sigma'', \hat{v}\}$ and $\hat{E} = E \cup \{vv' : v \in V\} \cup \{\hat{v}s, \hat{v}s', s'\sigma'' : s \in S\} \cup \{t''\bar{t}'', t''\sigma'' : t, \bar{t} \in T\}$. This is illustrated in Figure 5a.4.

We first show that $W_a = V \setminus \{a\}$ is a district of $\hat{G}$ for all $a \neq \hat{v}$ (note that $W_a \cap N(a; \hat{G}) = N(a; \hat{G})$). Necessarily one of the following holds.

- $a = s \in S$.

  Then $N(s; \hat{G}) = \{\hat{v}, s'\} \cup N(s; G)$ is dominated by the independent set $\{\sigma''\} \cup N(s; G)'$.

- $a = s' \in S'$.

  Then $N(s'; \hat{G}) = \{\hat{v}, s, \sigma''\}$ is dominated by the independent set $\{\bar{s}, \bar{s}'\}$ where $\bar{s} \in S \setminus \{s\}$ (and so necessarily $s\bar{s} \in E$).

- $a = \sigma''$.
  Then $N(\sigma''; \hat{G}) = S' \cup T''$ is dominated by the independent set $\{\hat{v}\} \cup T'$.
- $a = t \in T$.
  Then $N(t; \hat{G}) = t' \cup N(t; G)$ is dominated by the independent set $\{t''\} \cup N(t; G)'$ (or alternatively $\{t'', \hat{v}\}$).
- $a = t' \in T'$.
  Then $N(t'; \hat{G}) = \{t, t''\}$ is dominated by the independent set $\{\bar{t}, \bar{t}''\}$ where $t\bar{t} \in E$. (Recall $G - S$ has no isolated vertices in a Constituency instance of complete type.)
- $a = t'' \in T''$.
  Then $N(t''; \hat{G}) = \{t', \sigma''\} \cup (T'' \setminus \{t''\})$ is dominated by the independent set $\{t\} \cup (V' \setminus \{t'\}) \cup S'$.

We now show that $W_{\hat{v}}$ is a district of $\hat{G}$ if and only if $S$ is a constituency of $G$. We remark that $W_{\hat{v}} \cap N(\hat{v}; \hat{G}) = S \cup S'$. If $W_{\hat{v}}$ is a district of $\hat{G}$, then $S \subseteq N(I \setminus N[\hat{v}; \hat{G}]; \hat{G})$ for some independent set $I$. Therefore $S \subseteq N(I \cap T; G)$, i.e. $S$ is a constituency of $G$. Conversely, if $S$ is a constituency of $G$, say $S \subseteq N(I; G)$ for some independent set $I$ of $G$, then $I \cup \{\sigma''\}$ is an independent set of $\hat{G}$ such that $S \cup S' \subseteq N(I \cup \{\sigma''\}; \hat{G})$, i.e. $W_{\hat{v}}$ is a district of $\hat{G}$.

□

## 5a.4 Reachability of the MIS network

### 5a.4.1 The MIS network

By identifying a configuration $x \in \{0,1\}^V$ with its support $\mathbf{1}(x)$, one can interpret the MIS algorithm as sequential updates of a particular Boolean network. The **MIS network** on $G$, denoted as $\mathsf{M}(G)$ or simply $\mathsf{M}$ when the graph is clear from the context, is defined by

$$\begin{aligned}\mathsf{M}(x)_v &= \begin{cases} 0 & \text{if } \exists u \in N(v) : x_u = 1 \\ 1 & \text{if } \forall u \in N(v) : x_u = 0 \end{cases} \\ &= \bigwedge_{u \sim v} \neg x_u,\end{aligned}$$

with $\mathsf{M}(x)_v = 1$ if $N(v) = \emptyset$. We then have $\mathrm{Fix}(\mathsf{M}(G)) = \mathrm{M}(G)$ [152, 153].

The MIS algorithm then begins with the all-zero configuration, updates the state of every vertex in order, and leads to a configuration whose support is a maximal independent set. In the language of Boolean networks:

- $x = 0$;
- $w$ is a permutation of $V$;
- $y = \mathsf{M}^w(x) \in \mathrm{M}(G)$.

The pivotal role of constituencies for the MIS network can be explained by the equivalence below (its proof is easy and hence omitted). For a set of vertices $S \subseteq V$,

$S$ is a constituency of $G$ if and only if there exists a fixed point $y \in \mathrm{M}(G)$ such that $y_S = 0$.

### 5a.4.2 Universal configurations

In this chapter, we are interested in removing the constraint on the initial configuration $x$. This in turn will lead to constraints on the word $w$, as we shall see in the sequel. For now, in this section, we are interested in initial configurations $x$ that can lead to any MIS $y$.

Say a configuration $x$ is **$\mathsf{F}$-universal** if every fixed point of $\mathsf{F}$ is reachable from $x$, i.e. $x \mapsto_{\mathsf{F}} z$ for all $z \in \mathrm{Fix}(\mathsf{F})$. Clearly, the all-zero configuration is $\mathsf{M}(G)$-universal, as one can reach any MIS from the empty set. In fact, those fixed points can be reached by a geodesic. We now classify the universal configurations for the MIS network, which actually also allow to reach all fixed points by a geodesic. Since the classification is based on constituencies, the problem of deciding whether a configuration is universal is **coNP**-complete.

M-Universal Configuration

**Input:** A graph $G$ and a configuration $x$.

**Question:** Is $x$ an $\mathsf{M}(G)$-universal configuration?

**Theorem 5a.7** ([50]). M-Universal Configuration is **coNP**-complete.

We first characterise the configurations $y$ that are reachable from a given configuration $x$. For any configuration $x$ on $G$, we denote the collection of connected components of $G[\mathbf{1}(x)]$ as $\mathcal{C}(x)$. Before giving the full statement of the result, we provide some intuition. Suppose $y$ is reachable from $x$; we show that $y$ must satisfy two conditions. First, $y$ cannot "create an edge": if $[w]$ intersects an edge of $G[\mathbf{1}(x)]$, then it will destroy it. Therefore, any edge in $G[\mathbf{1}(y)]$ must be an (untouched) edge of $G[\mathbf{1}(x)]$. Second, $y$ cannot "empty out" a connected component: in order to update a vertex $v$ from $x_v = y_v^{a-1} = 1$ to $y_v = y_v^a = 0$, there must be a neighbour $a$ of $v$ such that $y_v^{a-1} = 1$. Therefore, for any $C \in \mathcal{C}$, $y_C \neq 0$.

Proposition 5a.8 then shows that these two conditions are indeed sufficient for reachability, and in fact for reachability by a geodesic.

**Proposition 5a.8** (Reachability for the MIS network[50]). Let $G$ be a graph and $x, y$ be two configurations on $G$. The following are equivalent:

1. $x \mapsto_{\mathsf{M}} y$;
2. $x \overset{\text{geo}}{\longmapsto}_{\mathsf{M}} y$;
3. every edge in $G[\mathbf{1}(y)]$ is an edge in $G[\mathbf{1}(x)]$ and $y_C \neq 0$ for any $C \in \mathcal{C}(x)$.

**Corollary 5a.9** ([50]). The configuration $x$ is $\mathsf{M}(G)$-universal if and only if every $C \in \mathcal{C}(x)$ is a non-constituency of $G$.

In particular, the all-zero and all-one configurations are $\mathsf{M}$-universal for all graphs.

Another consequence of Proposition 5a.8 is that any initial configuration can reach a fixed point via a geodesic.

**Corollary 5a.10** ([50]). For any configuration $x$, there exists $y \in \mathrm{M}(G)$ such that $x \overset{\mathrm{geo}}{\longmapsto}_{\mathsf{M}} y$.

## 5a.5 Words fixing the MIS network

We now focus on words fixing the MIS network. As we shall prove later, every graph $G$ has a fixing word. Whether a word $w$ fixes the MIS network does not only depend on the set $[w]$ of vertices it visits. Indeed, as seen in Example 5a.1 for the graph $P_3$, the word $w = abc$ does not fix $\mathsf{M}$, while $w = acb$ does fix $\mathsf{M}$. We define Fixing Word to be the decision problem asking, for an instance $(G, w)$, whether $w$ fixes $\mathsf{M}(G)$.

Fixing Word

**Input:** A graph $G = (V, E)$ and a word $w$.

**Question:** Does $w$ fix $\mathsf{M}(G)$?

**Theorem 5a.11.** Fixing Word is **coNP**-complete.

Fixing Word is in **coNP**; the certificate being a configuration $x$ such that $\mathsf{M}^w(x) \notin \mathrm{M}(G)$. Fixing Word is **coNP**-complete, even when restricted to permutations – which directly implies Theorem 5a.11 (we discuss this in in Section 5a.5.3).

### 5a.5.1 Prefixing and suffixing words

The seminal observation is that if $G$ is a graph, and $w$ is a permutation of $V$, then $ww$ fixes $\mathsf{M}(G)$: for any initial configuration $x$, $\mathsf{M}^w(x) \in \mathrm{I}(G)$; then for any $y \in \mathrm{I}(G)$, $\mathsf{M}^w(y) \in \mathrm{M}(G)$. We shall not prove this claim now, as we will prove stronger results in the sequel (see Propositions 5a.13 and 5a.15).

Following the seminal observation above, we say that $w^{\mathrm{p}}$ **prefixes** $\mathsf{M}(G)$ if $\mathsf{M}^{w^{\mathrm{p}}}(x) \in \mathrm{I}(G)$ for all $x \in \{0,1\}^V$, and that $w^{\mathrm{s}}$ **suffixes** $\mathsf{M}(G)$ if $\mathsf{M}^{w^{\mathrm{s}}}(y) \in \mathrm{M}(G)$ for all $y \in \mathrm{I}(G)$. In that case, for any word $\omega$, $w^{\mathrm{p}}\omega$ also prefixes $\mathsf{M}(G)$ and $\omega w^{\mathrm{s}}$ also suffixes $\mathsf{M}(G)$. Clearly, if $w = w^{\mathrm{p}} w^{\mathrm{s}}$, where $w^{\mathrm{p}}$ prefixes $\mathsf{M}(G)$ and $w^{\mathrm{s}}$ suffixes $\mathsf{M}(G)$, then $w$ fixes $\mathsf{M}(G)$. We can be more general, as shown below.

**Proposition 5a.12** ([50]). If $w = w_{1:l}$ where $w_{1:a}$ prefixes $\mathsf{M}(G)$, $w_{b:l}$ suffixes $\mathsf{M}(G)$, and $[w_{b:a}]$ is an independent set of $G$ for some $0 \leq a, b \leq l$, then $w$ fixes $\mathsf{M}(G)$.

We now characterise the words that prefix (or suffix) the MIS network. Interestingly, those properties depend only on $[w]$. Also, while deciding whether a word prefixes the MIS network is computationally tractable, deciding whether a word suffixes the MIS network is computationally hard as it is based on the Non-District problem.

**Proposition 5a.13** ([50])**.** Let $G$ be a graph. Then the word $w$ prefixes $\mathsf{M}(G)$ if and only if $[w]$ is a vertex cover of $G$.

Prefixing Word

| | |
|---|---|
| **Input:** | A graph $G = (V, E)$ and a word $w$. |
| **Question:** | Does $w$ prefix $\mathsf{M}(G)$? |

**Corollary 5a.14.** Prefixing Word is in **P**.

**Proposition 5a.15** ([50])**.** Let $G$ be a graph. Then the word $w$ suffixes $\mathsf{M}(G)$ if and only if $[w]$ is a non-district of $G$.

Suffixing Word

| | |
|---|---|
| **Input:** | A graph $G = (V, E)$ and a word $w$. |
| **Question:** | Does $w$ suffix $\mathsf{M}(G)$? |

**Corollary 5a.16** (following from Theorem 5a.4)**.** Suffixing Word is **coNP**-complete.

### 5a.5.2 Fixing sets

Some graphs have fixing words that do not visit all vertices. For instance, if $G = K_n$ is the complete graph with vertices $v_1, \dots, v_n$, then it is easily shown that $w = v_1 \dots v_{n-1}$ is a fixing word for the MIS network. In general, we say that a set $S$ of vertices of $G$ is a **fixing set** of $G$ if there exists a word $w$ with $[w] = S$ that fixes $\mathsf{M}(G)$.

We first characterise the fixing sets of graphs. Interestingly, those are the same sets $S$ such that $ww$ is a fixing word of $\mathsf{M}(G)$ for any permutation $w$ of $S$.

**Proposition 5a.17** ([50])**.** Let $S$ be a subset of vertices of $G$. The following are equivalent.

1. $S$ is a fixing set of $\mathsf{M}(G)$, i.e. there exists a fixing word $w$ of $\mathsf{M}(G)$ with $[w] = S$.
2. For all words $w^{\mathrm{p}}$, $w^{\mathrm{s}}$ such that $[w^{\mathrm{p}}] = [w^{\mathrm{s}}] = S$, the word $w^{\mathrm{p}}w^{\mathrm{s}}$ fixes $\mathsf{M}(G)$.
3. $S$ is a vertex cover and a non-district.

The Fixing Set problem asks, given a graph $G$ and a set of vertices $S$, if $S$ is a fixing set of $G$. In other words, it asks whether the vertices outside of $S$ can be skipped by some fixing word.

Fixing Set

| | |
|---|---|
| **Input:** | A graph $G = (V, E)$ and a set $S \subseteq V$. |
| **Question:** | Is $S$ a fixing set of $\mathsf{M}(G)$? |

**Theorem 5a.18** ([50])**.** Fixing Set is **coNP**-complete.

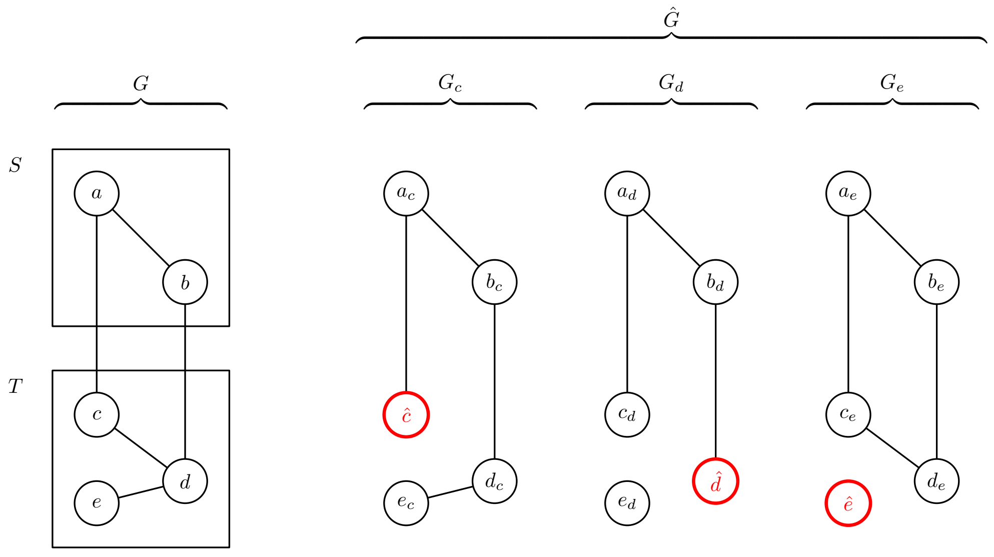


Figure 5a.5: Example reduction from a no-instance of Non-District $(G, S)$ to the corresponding no-instance of Fixing Set $(\hat{G}, \hat{S})$, with $\hat{S} = V_c \cup V_d \cup V_e$.

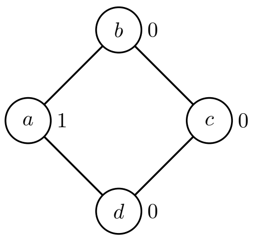


Figure 5a.6: $C_4$ is a no-instance of the Non-Trivial Non-District problem, and hence also a no-instance of Non-Trivial Fixing Set. For any word $w$ with $[w] = \{b, c, d\}$, $\mathsf{M}^w(1000) = 1000 \notin \mathrm{M}(C_4)$. By symmetry, no set of three vertices is a fixing set for $\mathsf{M}(C_4)$.

Clearly, if $S$ is a fixing set of $\mathsf{M}(G)$, then every superset of $S$ is also a fixing set. Moreover, every graph $G$ has a trivial fixing set, namely $V$. The Non-Trivial Fixing Set asks whether $G$ has any other fixing set. Equivalently, it asks whether any vertex can be skipped by a fixing word.

Non-Trivial Fixing Set

**Input:** A graph $G$.

**Question:** Does there exist a fixing set $S \neq V$ of $G$?

**Theorem 5a.19** ([50]). Non-Trivial Fixing Set and Non-Trivial Non-District are the same problem (have exactly the same yes- and no-instances), and consequently Non-Trivial Fixing Set is **coNP**-complete.

### 5a.5.3 Permises

The MIS algorithm doesn't use any word $w$ to update the state of each vertex, but instead restricts itself to $w$ being a permutation of $V$. As such, we now focus on permutations and we call a permutation of $V$ that fixes $\mathsf{M}(G)$ a **permis** of $G$. The Permis decision problem is equivalent to the Fixing Word problem, restricted to permutations.

PERMIS

**Input:** An undirected graph $G = (V, E)$ and a permutation $w$ of $V$.

**Question:** Is $w$ a permis of $G$?

Let $w$ be a permutation of $V$, then $w$ naturally induces a linear order on $V$, whereby $w_i \succ w_j$ whenever $i < j$, i.e. $w_i$ is updated before $w_j$. Then consider the orientation of $G$ induced by $w$: $G^w = (V, E^w)$ with $E^w = \{(u, v) : uv \in E, u \succ v\}$. We see that $G^w$ is acyclic, and that conversely any acyclic orientation of $G$ is given by some $G^w$. A simple application of [162, Theorem 1] shows that if $w, w'$ are two permutations of $V$ such that $G^w = G^{w'}$, then $w$ is a permis if and only if $w'$ is a permis.

We say that the vertex $v$ is **covered** by $w$ if for every $x \in \{0,1\}^V$, $y_{N[v]} \neq 0$, where $y = \mathsf{M}^w(x)$. Thus, $w$ is a permis if and only if $w$ covers all vertices.

COVERED VERTEX

**Input:** A graph $G = (V, E)$, a permutation $w$ of $V$ and a vertex $v \in V$.

**Question:** Is $v$ covered by $w$?

We now give a sufficient condition for a vertex to be covered. Let $G$ be a graph, $H$ be an orientation of $G$, and let $t$ and $v$ be vertices of $G$. We say $t$ is **transitive** if for all $a, b \in V$, $t \to a \to b$ implies $t \to b$ in $G^w$. We say $v$ is **near-transitive** if there exists a transitive vertex $t$ such that $N[t; G] \subseteq N[v; G]$. The following two results were obtained by leveraging the complexity of from NON-CONSTITUENCY.

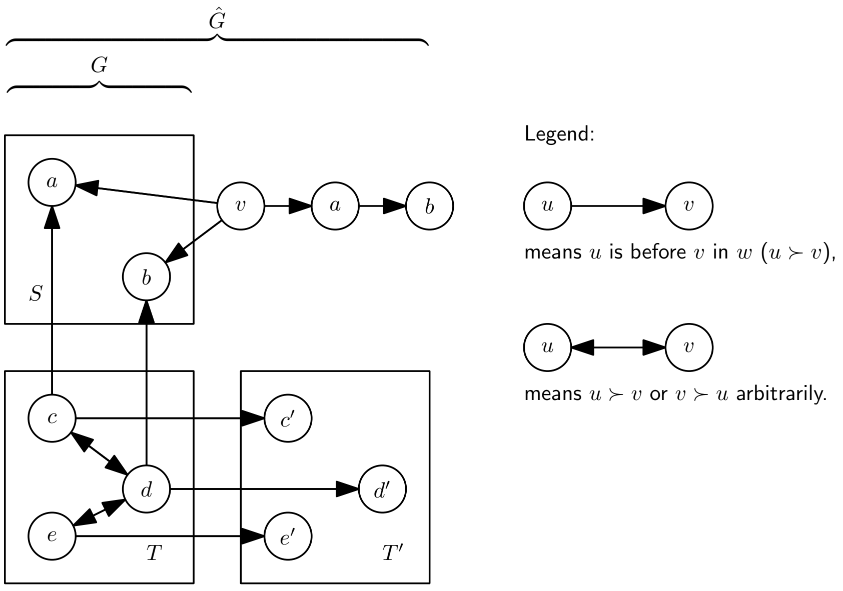
**Theorem 5a.20** ([50]). PERMIS is **coNP**-complete.

*Proof.* Membership of **coNP** is known: the no-certificate is a configuration $x$ such that $y = \mathsf{M}^w(x) \notin \mathrm{M}(G)$.

The hardness proof is by reduction from NON-CONSTITUENCY, which is **coNP**-complete by Corollary 5a.3. Let $(G, S)$ be an instance of NON-CONSTITUENCY of empty type and construct the instance $(\hat{G}, w)$ of PERMIS as follows. Let $T = V \setminus S$ and $T' = \{t' : t \in T\}$ be a copy of $T$. Then let $\hat{G}$ be the graph with vertex set $\hat{V} = \{v, a, b\} \cup V \cup T'$, and with edges $\hat{E} = E \cup \{sv : s \in S\} \cup \{va, ab\} \cup \{tt' : t \in T\}$. Let $w$ be a permutation of $\hat{V}$ such that $v \succ a \succ b \succ T \succ T' \succ S$. This is illustrated in Figure 5a.7.

We claim that $w$ is a permis of $\hat{G}$ if and only if $S$ is not a constituency of $G$. Firstly, the vertices in $S \cup T' \cup \{b\}$ are all transitive and hence the vertices in $T \cup \{a\}$ are near-transitive. Therefore, $w$ is a permis if and only if $v$ is covered. We prove that $v$ is covered if and only if $S$ is not a constituency of $G$.

If $S$ is a constituency of $G$, then let $I \subseteq T$ be a maximal independent set of $G$ (and hence an independent set of $\hat{G}$ as well) such that $S \subseteq N(I)$. Let $x = \chi(I \cup \{a, b\})$. Then $y_v = 0$ (because $x_a = 1$), $y_a = 0$ (because $x_b = 1$), $y_I = 1$ and $y_S = 0$ (because



$\hat{G}$
$G$
$a$
$v$
$a$
$b$
$b$
$S$
$c$
$d$
$e$
$T$
$c'$
$d'$
$e'$
$T'$
Legend:
$u$
$v$
means $u$ is before $v$ in $w$ ($u \succ v$),
$u$
$v$
means $u \succ v$ or $v \succ u$ arbitrarily.

Figure 5a.7: Illustration of the reduction from Non-Constituency to Permis.

for any vertex $u$, if $x_u = 1$ and $x_{N(u)} = 0$, then $y_u = 1$ and $y_{N(u)} = 0$). Thus $y_{N[v]} = 0$.

Conversely, if $y_{N[v]} = 0$, then for any $s \in S$, $y_{N(s)} \neq 0$. Since $y_v = 0$, there is $t \in T$ such that $ts \in E$ and $y_t = 1$. Therefore, the set $\mathbf{1}(y) \cap T$ is an independent set that dominates $S$, i.e. $S$ is a constituency of $G$. □

**Theorem 5a.21** ([50]). Covered Vertex is **coNP**-complete.

## 5a.6 Permissible and non-permissible graphs

We say that $G$ is **permissible** if it has a permis. As we shall see, not all graphs are permissible. In this subsection, we exhibit permissible and non-permissible graphs, and we prove that deciding whether a graph is permissible is computationally hard.

We classified (non-)permissible graphs by computer search, using `nauty`'s `geng` utility [163] to exhaustively generate connected graphs up to 9 vertices. Full results are available at https://github.com/dave-ck/MISMax/. Here are some highlights. Of 273194 connected graphs on at most nine vertices, only 432 are non-permissible; the heptagon $C_7$, 13 8-vertex graphs (including the perfect graph shown in Figure 5a.8), and 418 9-vertex graphs. The Petersen graph is also non-permissible.

We prove the graph in Figure 5a.8 is perfect as follows. First note that four vertices in the graph have degree 3 and four vertices in the graph have degree 5. The absence of an induced $C_5$ can be verified manually. There is no induced $C_7$: any subgraph on seven vertices includes at least one vertex formerly of degree 5 hence of degree at least 4 in the induced subgraph. Similarly, there is no induced $\overline{C_7}$; any subgraph on seven vertices includes at least one vertex formerly of degree 3 and hence of degree at most 3 in the induced subgraph ($\overline{C_7}$ is 4-regular).

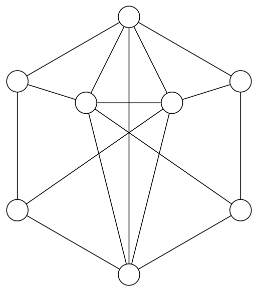


Figure 5a.8: An 8-vertex perfect graph with no permis.

### 5a.6.1 Permissible graphs

We now exhibit large classes of permissible graphs.

An orientation of $G$ is **transitive** (**near-transitive**, respectively) if all the vertices are transitive (near-transitive, respectively). Any transitive orientation is necessarily acyclic. A graph that admits a transitive orientation is called a **comparability graph**. The following are comparability graphs: complete graphs, bipartite graphs, permutation graphs, cographs, and interval graphs. Accordingly, we say that a graph that admits a near-transitive orientation is a **near-comparability graph**.

**Proposition 5a.22** ([50])**.** All near-comparability graphs are permissible.

We now give a characterisation of near-comparability graphs below. Recall that the vertex $m$ is a benjamin of $G$ if there is no vertex $v$ with $N^{\mathrm{in}}[v] \subset N^{\mathrm{in}}[m]$ and that $G_{\mathrm{B}}$ is the subgraph of $G$ induced by its benjamins.

**Theorem 5a.23.** reduction from NON-CONSTITUENCY to PERMIS Let $G$ be a graph. The following are equivalent:

1. $G$ is a near-comparability graph, i.e. it admits a near-transitive orientation;

2. $G$ admits a near-transitive acyclic orientation;

3. $G_{\mathrm{B}}$ is a comparability graph.

Recognising comparability graphs can be done in polynomial time; see [164] and references therein. In fact, the algorithm in [164] not only decides whether a graph is a comparability graph, but it also produces a transitive orientation if such exists. In view of Theorem 5a.23, applying that algorithm to $G_{\mathrm{B}}$ not only decides whether a graph is a near-comparability graph, but it also produces a near-transitive orientation if one exists.

Any induced subgraph of a comparability graph is a comparability graph. However, as we shall prove below, any graph is the induced subgraph of some near-comparability graph. Thus, Proposition 5a.22 shows that every graph is the induced subgraph of a permissible graph. This entails that the class of permissible graphs is not hereditary, i.e. it is impossible to characterize permissible graphs by some forbidden induced subgraphs.

**Corollary 5a.24** ([50])**.** For every graph $G$, there exists a near-transitive (and hence permissible) graph $H$ such that $G$ is an induced subgraph of $H$.

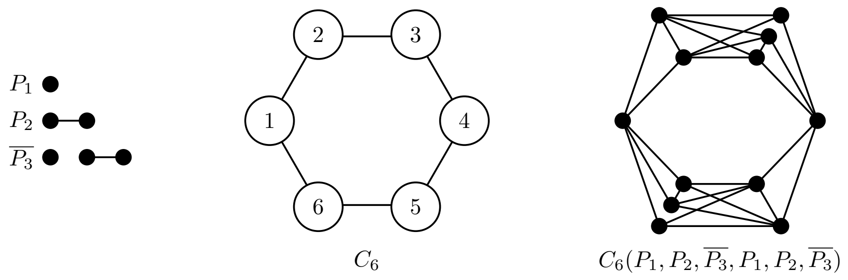


Figure 5a.9: Illustration of the composition operation.

We now introduce an operation on graphs, that we call **graph composition**, that preserves permissibility. Let $H$ be an $n$-vertex graph, $G_1, \ldots, G_n$ other graphs, then the composition $H(G_1, \ldots, G_n)$ is obtained by replacing each vertex $h$ of $H$ by the graph $G_h$, and whenever $hh' \in E(H)$, adding all edges between $G_h$ and $G_{h'}$. More formally, we have

$$V(G) = \{v_h : h \in V(H), v \in V(G_h)\},$$
$$E(G) = \{v_h v'_{h'} : hh' \in E(H), v \in V(G_h), v' \in V(G_{h'})\} \cup \{v_h v'_h : h \in V(H), vv' \in E(G_h)\}.$$

This is illustrated in Figure 5a.9.

This construction includes for instance the disjoint union of two graphs: $G_1 \cup G_2 = \bar{K}_2(G_1, G_2)$ and the join of two graphs: $K_2(G_1, G_2)$. In the special case where only a single vertex $b$ is replaced by the graph $G_b$, we use the notation

$$H(b, G_b) = H(K_1, \ldots, K_1, G_b, K_1, \ldots, K_1).$$

This special case includes adding an open twin (a new vertex $b'$ with $N(b') = N(b)$): $H(b, \bar{K}_2)$ and adding a closed twin ($N[b'] = N[b]$): $H(b, K_2)$. In fact, any composition can be obtained by repeatedly replacing a single vertex.

**Lemma 5a.25** ([50])**.** Let $G = H(G_1, \ldots, G_n)$ be a graph composition, where $V(H) = \{1, \ldots, n\}$. For all $0 \leq i \leq n$, let $G^i$ be defined as $G^0 = H$ and $G^i = G^{i-1}(i, G_i)$. Then $G^n = G$.

**Proposition 5a.26** ([50])**.** If $G = H(G_1, \ldots, G_n)$, where each of $H, G_1, \ldots, G_n$ is permissible, then $G$ is permissible.

### 5a.6.2 Non-permissible graphs

We now exhibit classes of non-permissible graphs. As mentioned earlier, the smallest non-permissible graph is the heptagon; in fact, any odd hole with at least seven vertices is non-permissible.

**Proposition 5a.27** ([50])**.** For all $k \geq 3$, the odd hole $C_{2k+1}$ is not permissible.

We now give two ways to construct larger non-permissible graphs.

Recall that a set of vertices $S$ is tethered if there is an edge $st$ between any $s \in S$ and any $t \in T = N(S) \setminus S$.

**Proposition 5a.28** ([50])**.** Let $G$ be a graph. If $G$ has a tethered set of vertices $S$ such that $G[S]$ has no permis, then $G$ has no permis.

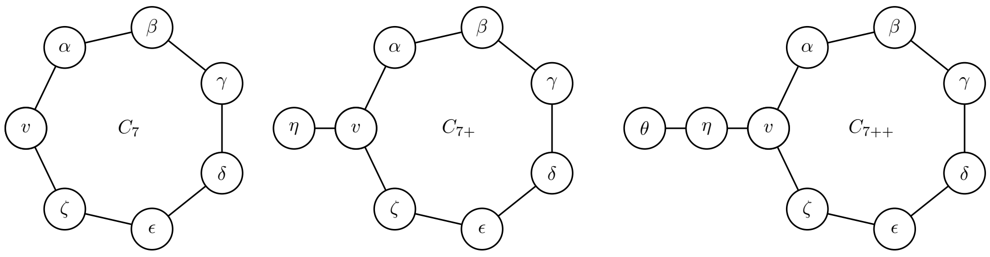


Figure 5a.10: Graphs $C_7$ (non-permissible), $C_{7+}$ (permissible), and $C_{7++}$ (non-permissible).

Propositions 5a.27 and 5a.28 yield perhaps the second simplest class of non-permissible graphs. The **wheel graph** is $W_{n+1} = K_2(C_n, K_1)$.

**Corollary 5a.29.** For all $k \geq 3$, the wheel graph $W_{2k+2}$ is not permissible.

Clearly, a graph is permissible if and only if all its connected components are permissible. In particular, for any $G$, the union $H = G \cup C_7$ is not permissible, but is disconnected. An interesting consequence of Proposition 5a.28 is that permissibility of a connected graph cannot be decided by focusing on an induced subgraph, even if the latter has all but seven vertices of the original graph. Indeed, for any graph $G$, the join $H' = K_2(C_7, G)$ is not permissible, since the heptagon is tethered in $H'$.

**Corollary 5a.30.** Let $G$ be a graph. Then there exists a connected non-permissible graph $H'$ such that $G$ is an induced subgraph of $H'$.

Second, and unsurprisingly, we can construct larger non-permissible graphs by using a constituency.

**Proposition 5a.31** ([50])**.** Let $G$ be a graph. If there exists $A \subseteq V$ such that $G[A]$ is not permissible and $S = N(A) \setminus A$ is a constituency of $G - A$, then $G$ is not permissible.

For all $k \geq 3$ the odd hole $C_{2k+1}$ is non-permissible. Consider the graph $C_{2k+1+}$ by adding a vertex of degree one to $C_{2k+1}$; as we shall see later it is permissible. Now add another vertex of degree one to the tail of $C_{2k+1+}$ to obtain $C_{2k+1++}$. The graphs $C_7$, $C_{7+}$ and $C_{7++}$ are illustrated in Figure 5a.10. In $C_{7++}$, the vertex $S = \{\eta\}$ is a constituency and is the neighbourhood of the heptagon; therefore $C_{7++}$ is not permissible. Obviously, this reasoning applies to all larger $C_{2k+1++}$ as well.

**Corollary 5a.32.** For all $k \geq 3$, $C_{2k+1++}$ is not permissible.

### 5a.6.3 The PERMISSIBLE decision problem

We now explore the complexity of deciding whether a graph is permissible.

PERMISSIBLE

**Input:** A graph $G$.

**Question:** Is $G$ permissible?

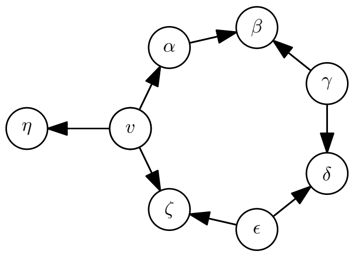


Figure 5a.11: The permis of $C_{7+}$.

As mentioned above, $C_{7+}$ is permissible; the proof of Lemma 5a.33 below can be generalised to show that $C_{2k+1+}$ is permissible for all $k \geq 3$.

**Lemma 5a.33** ([50])**.** Any $w$ such that ${C_{7+}}^w$ is given on Figure 5a.11 is a permis of $C_{7+}$.

**Theorem 5a.34** ([50])**.** PERMISSIBLE is **coNP**-hard.

## 5a.7 Conclusions and future work

In this chapter, we have considered the generalisation of the MIS algorithm to allow for any initial configuration and to use update words that are not necessarily permutations. We have defined many decision problems with respect to this generalisation, such as:

- Given $G$ and a configuration $x$, can $x$ reach all MIS?
- Given $G$ and a word $w$, does $w$ fix $\mathsf{M}(G)$?
- Given $G$ and a set of vertices $S$, can we fix $\mathsf{M}(G)$ by only updating $S$?
- Given $G$, is there a word fixing $\mathsf{M}(G)$ that skips a vertex?
- Given $G$ and a permutation $w$, does $w$ fix $\mathsf{M}(G)$ (i.e. is $w$ a permis of $G$)?
- Given $G$, does $G$ have a permis?

Even though every graph has a fixing word that guarantees terminating at a MIS regardless of the initial configuration, all the decision problems about the MIS algorithm in this chapter are computationally hard. Additionally, we exhibit broad classes of graphs with and without permises, and relate these to existing graph classes. We introduce the class of near-comparability (a strict superclass of comparability graphs, which themselves encompass interval graphs and bipartite graphs, among others) and show that all near-comparability graphs are permissible.

This work can be extended in several ways. We give three potential avenues below.

**Graph classes.** Since our problems are **NP**- or **coNP**-hard for the class of all graphs, it is natural to examine the complexity of those problems when we restrict ourselves to particular graph classes. The main tool for reductions is the CONSTITUENCY decision problem. However, the reductions used in this chapter did not preserve certain graph classes. For instance, CONSTITUENCY remains **NP**-complete even for bipartite graphs, while PERMISSIBLE is trivial for comparability graphs.

**Minimum length of a fixing word.** We have investigated the existence of fixing words, but not their lengths. From our results on prefixing and suffixing words, we get an upper bound on the minimum length of a fixing word: $a+b-1$, where $a$ is the minimum size of a non-district and $b$ is the minimum size of a vertex cover. We conjecture that the problem of determining the minimum length of a fixing word is computationally hard. The analogous problem when we can only start at the all-zero configuration is obviously **NP**-hard, as this amounts to determining the minimum size of a maximal independent set.

**Permises with bounded diameter.** Let $w$ be a permutation of $V$, and for any vertex $v$, let $\delta(v)$ denote the maximum length of a path terminating at $v$ in $G^w$. Let $C_i = \{v \in V : \delta(v) = i\}$, then $V = C_0 \cup \cdots \cup C_d$, where $d$ is the diameter of $w$. Instead of updating the vertices sequentially according to $w$, one can update all the vertices in $C_i$ at once, thus only requiring $d+1$ time steps. As such, the diameter of a permis $w$ measures the time it takes to fix the MIS network when we allow for some amount of synchronicity. If $\Delta$ is the maximum degree of $G$, then $G$ always has a permutation of diameter $\Delta$: partition $V$ into colour classes $C_0, \dots, C_\Delta$ (since the chromatic number is at most $\Delta + 1$), then $C_0 \succ C_1 \succ \cdots \succ C_\Delta$. This is best possible if $G$ is complete, for instance. We therefore ask: if $G$ has a permis, then does it have a permis of diameter bounded by a function of $\Delta$?

Chapter 5b

# Better Late, then? Delaying connections in temporal graphs.

*This chapter is based on joint work [51] with Anouk Sommer, a student at the Karlsruhe Institute of Technology in Germany. Most of the results presented here were discovered collaboratively in summer 2024 during a research internship supported by the Deutscher Akademischer Austauschdienst (DAAD) with the title* Schnell aber spät: breaking Deutsche Bahn even more with graph theory.

## 5b.1 Introduction

In the first half of 2024, punctuality of Deutsche Bahn's long distance trains was 62.7% [165]. Disruptions to train networks often result in passengers arriving later than planned or not at all. Whenever a train is late and the passengers of this train would miss a connecting train, there are two choices: either, the second train departs on time, meaning that the passengers of the first train miss their connection, or the second train waits, meaning that the passengers can make the connection, at the cost of this train now also being late. The problem of deciding whether (and by how much) such services should wait is the Delay Management problem, well studied in Operations Research.

Separately, the field of temporal graph theory provides a general, rigorous mathematical framework with which to investigate the complexity due to the intrinsically dynamic properties of certain real-world networks. Briefly, a temporal graph is one whose edge set changes over time. Much work has been devoted to *modification* problems of the form "Given a temporal graph $\mathcal{G}$, apply some (delaying or other) operations to satisfy some reachability property" (see Table 5b.1), but interestingly the problem of managing delays to ensure that specific passengers arrive at their destination on time has yet to be studied in this framework. fig. 5b.1 shows a simple example of a temporal graph illustrating such a scenario.

The present work aims to study the practically interesting problem of Delay Management through the lens of temporal graph theory. We introduce the decision problem DelayBetter (or simply DB) which asks, given a temporal graph and a col-

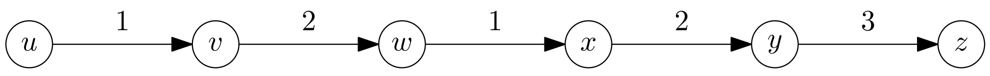


Figure 5b.1: A temporal graph on 6 vertices. Consider the case where passengers at each of $u, v$, and $w$ wish to travel to each of $x, y$, and $z$ respectively, arriving at or before time 4. Then delaying the edge from $w$ to $x$ by at least 2 is necessary for the two leftmost passengers to arrive on time, but entails that the passenger starting at $w$ cannot arrive at $z$ before time 5.

lection of passengers on its vertices, each with a desired destination and arrival time, whether it is feasible to delay some edges of the graph to satisfy each of the passengers. We also consider variants of the problem: PATH-DB, where passengers must be routed along specific edges prescribed in the input, $\delta$-DB, where each edge can be delayed by at most some fixed $\delta$, and $\delta$-PATH-DB combines both constraints. We present (parameterized) tractability and hardness results for these problems, including on structurally restricted graph classes.

| **Problem** | **Operation** | **Restriction** | **Reachability condition** | **Additional inputs** |
|---|---|---|---|---|
| REACHFAST [84] | Shift (+−) | N/A | $\forall x \in S : R_x = V$ | sources $S \subseteq V$, $\tau \in \mathbb{N}$ to be minimized |
| TRLP [85] | Shift (+−) | up to $\eta$ edges, by up to $\delta$ each | $\|R_x\| \geq k$ | designated source $x \in V$, $\eta, \delta, k \in \mathbb{N}$ |
| MINREACHDELAY [166] | Delay | up to $\eta$ time-edges by exactly $\delta$ each | $\|R_S\| \leq k$ | sources $S \subseteq V$, $\eta, \delta, k \in \mathbb{N}$ |
| MINREACH [72] | Delay | up to $\eta$ time-edges by up to $\delta$ each | $\|R_S\| \leq k$ | sources $S \subseteq V$, $\eta, \delta, k \in \mathbb{N}$ |
| MAXMINTARDIS (Chapter 3) | Choose time-labels | lifetime is $\tau$ | $\nexists S \subseteq V, \|S\| < k : R_S = V$ | $\tau, k \in \mathbb{N}$ |
| ($\delta$-)DELAYBETTER | Delay | (by up to $\delta$ per edge) | $(u, v, t) \in D \Rightarrow v \in R_u^t$ | $D \subseteq V \times V \times \mathbb{N}$ ($\delta \in \mathbb{N}$) |
| ($\delta$-)PATH DB | Delay | (by up to $\delta$ per edge) | as above along specified path | $D \subseteq V \times V \times \mathbb{N} \times 2^E$ ($\delta \in \mathbb{N}$) |

Table 5b.1: Comparison of our problems DELAYBETTER and PATH DELAYBETTER/PATH DB to problems in the literature. $R_u^t$ (resp. $R_S^t$) denotes the set of vertices reachable from vertex $u$ (resp. any vertex $s \in S$) by time-step $t$ (when $t$ is the lifetime of the temporal graph, it is omitted).

### 5b.1.1 Problem setting

We denote $[i, j]$ the integer interval $\{i, i + 1, \ldots, j\}$, and say a graph is *cubic* if all vertices in the graph have degree 3. We now give some definitions from temporal graph theory.

**Definition 5b.1** (Temporal graph, temporal path)**.** A *temporal graph* $\mathcal{G} = (G, \lambda)$ consists of a static graph $G$ (also called its *footprint*, denoted $\mathcal{G}_\downarrow$) and a *temporal* *assignment* $\lambda : E(G) \to \mathbb{N}$. The *lifetime* $\tau \in \mathbb{N}$ of a temporal graph $(G, \lambda)$ is the maximum time assigned to any edge by $\lambda$, and the *time-edges* of a temporal graph $\mathcal{E}(\mathcal{G})$ are $\{(u, v, \lambda(u, v)) | (u, v) \in E(G)\}$. A *temporal path* is a path in $\mathcal{G}_\downarrow$ whose edges have strictly increasing time-labels, and the *arrival time* of a temporal path is the time-label of its last edge.

We take this opportunity to note a more general definition of temporal graphs is often studied, where each edge may have multiple time labels (*non-simple* temporal graphs). Another variant applies a notion of temporal reachability which allows for the traversal of consecutive edges at nondecreasing (rather than strictly increasing) times (*non-strict* temporal paths). For a discussion of these different models (in the undirected setting only) we refer the interested reader to [78]. Hardness results from the simple setting generalize to the non-simple setting, and tractability for the non-simple setting may be applied to the simple setting. In the present work, we focus exclusively on strict temporal paths and simple (directed or undirected) temporal graphs.

**Definition 5b.2** (Delaying)**.** We say that a temporal assignment $\lambda'$ is a *delaying* of an assignment $\lambda$ if $\lambda'(e) \geq \lambda(e)$ for every $e$. If $\lambda'(e) = \lambda(e) + \delta$, we say that $e$ is delayed by $\delta$ in $\lambda'$, and that $\lambda'$ is a $\delta$-delaying of $\lambda$ if every $e$ is delayed by at most $\delta$ in $\lambda'$ (hence a $\delta$-delaying is also a $(\delta + 1)$-delaying).

We can now introduce our protagonists:

($\delta$-)DelayBetter
**Instance**: Temporal graph $\mathcal{G} = (G, \lambda)$, demands $D \subseteq V(G) \times V(G) \times \mathbb{N}$ (, $\delta \in \mathbb{N}$).
**Question**: Does there exist a ($\delta$-)delaying $\lambda'$ of $\lambda$ such that for each $(u, v, t) \in D$ there is a temporal path from $u$ to $v$ with arrival time at most $t$ in $(G, \lambda')$?

($\delta$-)Path DelayBetter
**Instance**: Temporal graph $\mathcal{G} = (G, \lambda)$, demands $D \subseteq V(G) \times V(G) \times \mathbb{N} \times 2^{E(G)}$ (, $\delta \in \mathbb{N}$).
**Question**: Does there exist a ($\delta$-)delaying $\lambda'$ of $\lambda$ such that for each $(u, v, t, P) \in D$ there is a temporal path from $u$ to $v$ in $(G, \lambda')$ with arrival time at most $t$ and footprint $P$?

We say a temporal graph $\mathcal{G}$ is *planar* if its footprint $\mathcal{G}_\downarrow$ is planar, and *directed* (resp. *undirected*) if $\mathcal{G}_\downarrow$ is directed (resp. undirected). We use the shorthand DB for our problems, referring to, for example, 3-DB, Path DB, or DB. For a demand $d$, we denote $d = (d_s, d_z, d_t)$. Restriction of, or parameterization by, the lifetime $\tau$ is often leveraged to obtain tractability of temporal graph problems. In our case, we denote by $T_{\text{init}}$ the *initial lifetime* (that of the temporal graph $\mathcal{G}$ before delays are applied), and by $T_{\max}$ the latest arrival time required by any single demand – that is, $\max_{d \in D} d_t$. We call $T_{\max}$ *final lifetime* because it upper-bounds the lifetime of the temporal graph $(G, \lambda')$ (after delays are applied): any time-edge delayed beyond $T_{\max}$ in some feasible solution could instead be delayed to $T_{\max}$ instead (or not at all), since it will not be used by any passengers. For the same reason, we may assume without loss that $T_{\text{init}}$ is at most $T_{\max}$.

### 5b.1.2 Related work

**Temporal Graphs.** As we touched on earlier, modifying (or choosing) $\lambda$ to optimize a notion of reachability is a well-studied problem in temporal graph theory. Broadly, problems in this paradigm may either aim to *worsen* or *improve* the input temporal graph's connectedness. Problems in the first category (including MinReach [72], MinReachDelay [166], and MaxMinTaRDiS from Chapter 3) are typically motivated by practical cases where spread is undesired, such as epidemics. In the case of transportation networks, where connectedness is desired, the second category (which contains ReachFast [84] and TRLP [85] ) is of greater relevance. Of course, if the delays are controlled by an adversary, the opposite motivation becomes relevant to each problem: is there any strategy for the adversary to disconnect a transporation network, or facilitate disease spread? A related, but slightly different perspective on delays in temporal graphs is explored in [167] and [19], who determine how robust against unforeseen delays a given temporal graph is with and without re-routing of the passengers, respectively.

There are also some high-level conceptual similarities between the Perfect Scheduling problem from Chapter 2 and the problems we consider here. In both cases, edge times are chosen to achieve a global property, namely the timely arrival of passengers or the timely payment of debts. A key difference between the two models is that in the former, every edge *must* be used (and some assets may sit unused at their starting vertex), whereas in the present model every passenger must move from their starting point to their destination, and some edges may be unused in the solution.

**Delay Management.** The Delay Management (DM) problem concerns itself with finding a good delaying strategy in a public transport network to minimize passenger inconvenience. Usually, this means minimizing the total passenger delay, but other objectives like simultaneously minimizing the number of delayed trains or the operational costs have also been studied. In the original problem, as introduced by [168], passengers stick with their initial routes (as in Path-DB); a popular variant of the problem allows passenger re-routing (as in DB) [169]. Both settings have since been the subject of much study, spanning both theory and practice.

On the theoretical side, different models and algorithmic approaches have been introduced over the years [170, 171, 172, 173, 174]. Due to modeling differences, studies of the computational complexity of different DM problem variants [175, 176, 177] do not necessarily yield results for our problems. In addition to minimizing an aggregate function (e.g., total weighted passenger delay [175, 176]) rather than asking whether some specific set of passenger demands can be satisfied (as we do), DM problems are commonly formalized using *event-activity networks* – which are more expressive than temporal graphs. For example, the definition of DM in [177] includes *headway constraints* (where two trains cannot use the same track segment simultaneously). Several interesting practically-motivated extensions are studied in this line of work, including a setting with slack times (trains may catch up on their delay), which makes the problem hard when the rail network is a line [176], and the incorporation of rolling-stock circulation into the problem [177] – though results in such settings do not straightforwardly translate into our model. Nonetheless, some results from these works can be adapted into the our setting; for example, Theorem 6.1 in [175] could be adapted to show that DelayBetter is NP-complete in the directed setting with $T_{\max} = 3$ (we strengthen this in theorem 5b.11). Also, all of these works consider a directed model (as is natural for rail networks), whereas our results are proven for both directed and undirected temporal graphs.

On the more practical side, there have been a number of case-studies and data-driven approaches to this problem [178, 179, 180]. In [181], a model for optimizing delays in rail and air travel combined is proposed, together with a European case study. For a more comprehensive overview of the work in delay management, we refer the reader to [182], [183], and [184]. A related area of research is the Timetabling Problem, which concerns itself with designing a timetable that is robust against delays. We refer the reader to [185] for an introduction.

### 5b.1.3 Our contribution

We introduce the problems ($\delta$-)DelayBetter and ($\delta$-)Path DB, presenting (to our knowledge for the first time) a temporal graph-theoretic approach to the well-studied Delay Management problem. On the positive side, we give a polynomial-time algorithm for ($\delta$-) Path-DB, and tractability for ($\delta$-)DelayBetter on trees as a corollary. Later, we leverage this algorithm to obtain a fixed-parameter tractable (fpt) algorithm parameterized by the number of demands and the size of the feedback edge set of the footprint graph. On the negative side, we establish that DelayBetter remains NP-complete on inputs with $T_{\max} = 2$ in both the directed and undirected setting (which entails that 1-DelayBetter is NP-complete under the same constraint). Moreover, we show that the problem remains hard on planar (directed or undirected) temporal graphs with $T_{\max} = 19$, even when $\delta = 10$. Our results provide a first insight into the structural restrictions which do (and do not) suffice to guarantee tractability of this natural problem. Proofs of statements marked ($*$) can be found in the appendix at the end of the paper.

## 5b.2 Preliminary Results

We begin with some basic results, the proofs of which may help to familiarize the reader with the behavior of our problems. We first establish a useful relation between

$\delta$-DB and DelayBetter. Clearly, DelayBetter is reducible to $\delta$-DelayBetter, by simply assigning a sufficiently large value to $\delta$ (e.g., the final lifetime $T_{\max}$ of the DelayBetter instance). Interestingly, the converse also holds:

**Lemma 5b.3.** For any $\delta \in \mathbb{N}$, $\delta$-DelayBetter is reducible in linear time to DelayBetter. If the input instance is planar (resp. has bounded final lifetime) then the same holds for the output.

*Proof.* We require different reductions for directed and undirected graphs. In both cases, substitute a gadget in place of each edge in the original instance, and increase the lifetime of the instance. Both constructions preserve planarity, and the DelayBetter-instance has final lifetime at most $2T_{\max} + 2\delta + 1$ (directed) or $T_{\max} + \delta + 2$ (undirected), where $T_{\max}$ is the final lifetime of the $\delta$-DelayBetter instance.

We first deal with the undirected case. We begin with a $\delta$-DelayBetter instance $((G, \lambda), D, \delta)$ having the property that every time is at time 3 or later (if necessary, this can be achieved by uniformly incrementing all times in the demands and in the temporal assignment by 2). We then create, for every time-edge $(u, v, t)$, a gadget on $3\delta+3$ new vertices $\{uv_t, \ldots, uv_{t+\delta}, u_1, \ldots, u_\delta, v_1, \ldots, v_\delta, u', v'\}$ and $\delta+1$ new demands $\{(uv_i, v', i + 1) : i \in [t, t + \delta]\}$, as shown in fig. 5b.2.

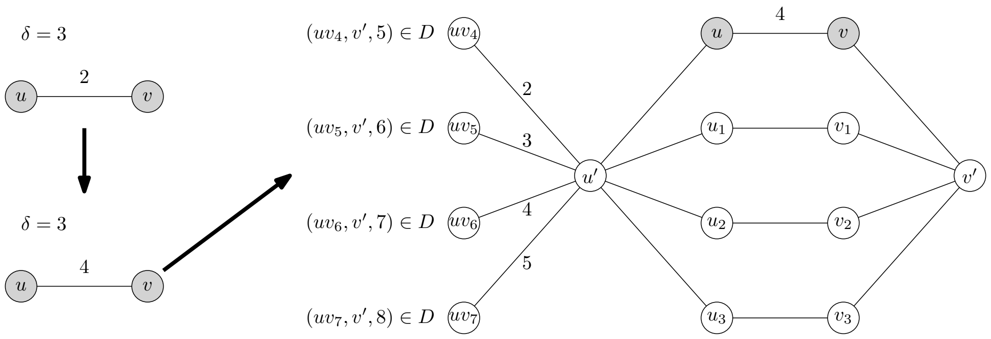


Figure 5b.2: A sketch of our reduction for undirected temporal graphs for a time-edge $(u, v, 2)$ in an instance with $\delta = 3$. For readability, edges assigned time 1 in the output instance are unlabeled.

Now each of the $\delta + 1$ demands into $v'$ must be routed through a different path. Because there are $\delta+1$ possible paths in total, some demand $(uv_i, v', t)$ must be routed through the edge $(u, v)$ – and this entails that $\lambda'(u, v) = i$, yielding the desired result since $i \in [t, t + \delta]$ by construction.

We now give the proof for the directed case. Given an instance $(\mathcal{G} = (G, \lambda), D, \delta)$ of $\delta$-DelayBetter, we produce an instance $(\mathcal{G}' = (G', \lambda'), D')$ of DelayBetter as follows. (For this proof, we use $\lambda'$ to refer to the initial assignment of the new instance, not the delaying of $\lambda$.) We first include $\{(u, v, 2t) | (u, v, t) \in D\}$ as demands, which we call *travelers*. Next, we replace every time-edge $(u, v)$ at time $t$ with the gadget pictured in Figure 5b.3, and add the demand $(u', v', 2t + 2\delta + 1)$. We call demands introduced in this step *hermits*, the edge $(u', u)$ that hermit's *trailhead*, and the edge $(u, uv)$ (resp. $(uv, v)$) a *first-half* (resp. *second-half*) edge. This concludes the construction.

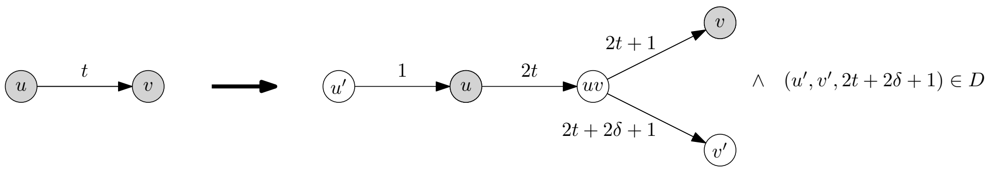


Figure 5b.3: A sketch of our reduction for directed temporal graphs.

Clearly, if the $\delta$-DB instance reduced from was a yes-instance, then the Delay-Better instance obtained is also a yes-instance: whenever some edge $(u, v)$ is delayed by some amount $x$ in the original instance, delay both $(u, uv)$ and $(uv, v)$ by $2x$. It remains to show the converse.

Let $\lambda^*$ be a solution to our modified problem with a pareto-optimal time-assignment of the edges (that is, one such that there is no other solution whose time-labels are all strictly smaller or equal to those under $\lambda^*$).

**Claim 5b.3.1.** Hermits leave early: if $e$ is a hermit's trailhead, then $\lambda^*(e) = 1$.

*Proof of claim:* The claim follows straightforwardly from pareto optimality of $\lambda^*$, and the fact that hermit trailheads are used only by the hermit, who can wait at the vertex $u$ instead of at $u'$. ■

**Claim 5b.3.2.** Under $\lambda^*$, every first-half edge (resp. second-half edge) is assigned an even (resp. odd) time.

*Proof of claim:* Suppose otherwise. We must deal with two cases:

We deal first with the case where the earliest edge violating this claim is a first-half at an odd time. Let $(u, uv)$ be the earliest first-half edge assigned an odd time (say, $t'$) under $\lambda^*$. By pareto-optimality of $\lambda^*$ is must be that assigning time $t' - 1$ to the edge $(u, uv)$ would stop some demand from being satisfied. This demand cannot be a hermit because of claim 5b.3.1 – so there must be some traveler arriving at $u$ at time $t' - 1$, a contradiction since our premise for this case was that $(u, uv)$ was the earliest edge violating the claim.

Suppose instead that the earliest offending edge is a second-half edge $(wu, u)$ assigned an even time (say, $t' - 1$). We again quickly find that this can only be due to the first-half edge $(w, wu)$ being used by a traveler at time $t' - 2$ - again reaching a contradiction, since this is a strictly earlier odd time assigned to a first-half edge. ■

**Claim 5b.3.3.** For each time-edge $(u, v, t)$ in the original instance, $\lambda^*(u, uv) \in [2t, 2t + 2\delta]$.

*Proof of claim:* By construction, there is a hermit demand $(u', v', 2t + 2\delta + 1)$. This hermit must use the edge $(u, uv)$ (since the new vertex $uv$ has no other incoming edges and $v'$ is only reachable from $uv$). The hermit must use this edge no earlier than time $2t$ (as this is its original time under $\lambda'$) and no later than time $2t + 2\delta$ (as the next edge in the temporal path must be at time $2t + 2\delta + 1$ exactly). ■

claim 5b.3.2 allows us to recover a time-labeling for our initial $\delta$-DelayBetter instance by assigning to each the edge $(u, v)$ the time $\frac{(u,uv)}{2}$ while preserving the temporal paths of travelers. claim 5b.3.3 entails that this time-labeling does not delay any edge by more than $\delta$, and the result follows. □

**Lemma 5b.4.** An instance of DelayBetter or Path DB (resp. $\delta$-DelayBetter or $\delta$-Path DB) may be reduced in polynomial time to an instance of the same problem with $T_{\max} \in \text{poly}(n)$ (resp. $T_{\max} \in \text{poly}(n+\delta)$).

*Proof.* Given an instance $(\mathcal{G}, D)$ of either problem, we identify the set of all *explicit* times (directly encoded in the input) as $T_{explicit} := \{d_t | d \in D\} \cup \{\lambda(e) | e \in E(\mathcal{G})\}$, where $d_t$ is the arrival time specified by $d$. Denote $|T_{explicit}|$ by $\alpha \leq |E| + |D|$ (this inequality is strict if any time appears explicitly more than once in $(\mathcal{G}, D)$). We then may sort $T_{explicit}$ into an ordered list of times $t_1 < t_2 < \ldots < t_\alpha$.

*Shrinking* of an interval $[t_i, t_j]$ to be of size $\ell$ consists in decrementing all times $t_j$ or greater in the original instance by $t_j - \ell - t_i \geq 0$. Thus, any edge (or demand) formerly at time $t_j$ is updated to be at time $t_i + \ell$. *Deleting* a time interval $[t_i, t_j]$ consists in shrinking that time interval to have size 0.

We first deal with DelayBetter and Path DB. Consider the integer intervals $[t_i, t_{i+1}]$. If any such interval has size greater than $|E|$, we may without loss shrink the interval to have size $|E|$ instead. No-instances of both problems are clearly preserved by the operation. Yes-instances are also preserved: only the relative order of times assigned to edges matters for a temporal path to exists, and any ordering achievable in the original instance is also achievable in the transformed instance since at most $|E|$ unique times are assigned under $\lambda'$ in total.

We now deal with $\delta$-DelayBetter. We identify the set of *relevant* times to be $T_{relevant} := \bigcup_{t \in T_{explicit}} : [t, t+\delta]$. Note that this set has cardinality at most $\delta \cdot (|E| + |D|)$, and that it contains all possible times used in any solution $\lambda'$. Hence we then may eliminate every time not in $T_{relevant}$ (by deleting at most $|E| + |D|$ intervals) and obtain an equisatisfiable instance with $T_{\max} \leq \delta \cdot \alpha$.

In both cases, the procedure clearly runs in time $\text{poly}(\log T_{\max} + |V(\mathcal{G})| + |D|)$, and we obtain the desired result. □

**Lemma 5b.5.** ($\delta$-)DelayBetter is contained in NP.

*Proof.* Given an instance $I = (\mathcal{G}, D)$ of ($\delta$-)DelayBetter and a corresponding solution, i.e., an assignment $\lambda'$ of time-labels (which can delay edges of the initial assignment $\lambda$), we can check in polynomial time whether $\lambda'$ is indeed a valid solution for $I$ as follows.

First, we need to check that the assignment $\lambda'$ actually represents valid delays (i.e., that no edge was moved to an earlier point in time). To do so, we check in $O(|\mathcal{E}|)$ whether for every $e \in \mathcal{E}$ we have $\lambda(e) \leq \lambda'(e)$ (for the case of $\delta$-DelayBetter, we also check that $\lambda(e) + \delta \leq \lambda'(e)$).

It remains to check the demands are met by the assignment. The earliest arrival time $\text{arr}_{u \to v}$ of any strict temporal path from $u$ to $v$ in the temporal graph $(G, \lambda')$ may be computed in polynomial time (see, e.g. [186]). It then suffices to verify, for each $(u, v, t) \in D$, that $\text{arr}_{u \to v} \leq t$, which can be done in polynomial time, and the result follows. □

## 5b.3 Tractability Results

We begin with a small positive result which can be obtained easily from prior work.

**Lemma 5b.6.** DelayBetter is solvable in polynomial time when $\lambda$ is the constant function $\mathbf{1}$ and all demands in $D$ have the same source.

*Proof.* We use the One Source Reach Fast algorithm from [84]: They show that the time-assignment of their algorithm computes, for a given source $v \in V$ and every remaining vertex $u \in V$, the individual minimum time that $v$ needs to reach $u$. If this computed minimum time is at most our demanded arrival time for all demands $(v, u, t) \in D$, then we have a YES-instance, otherwise we have a NO-instance. □

We now turn to the case where passenger demands fully prescribe the path they must be routed along, establishing tractability through a linear programming argument.

**Theorem 5b.7.** Path DelayBetter and $\delta$-Path DelayBetter are both in P.

*Proof.* Let $((G, \lambda), D)$ be an instance of Path DelayBetter (or, let $((G, \lambda), D, \delta)$ be an instance of $\delta$-Path-DelayBetter – the proof differs only in a few details).

We begin by introducing some notation. Our proof is for directed and undirected inputs – we shall use $uv$ to mean the edge $(u, v)$, but in the undirected case $uv = vu$ whereas in the directed case these are $uv \neq vu$. For a demand $d \in D$, we denote by $d_P$ the specified static path in $G$ from $d_s$ to $d_z$, and $d_f$ the final edge of $d_P$, which is incident to $d_z$ and must be at time $d_t$ or earlier to satisfy the demand. We also use $t_{uv}$ as shorthand for $\lambda(u, v)$, and $t'_{uv}$ for $\lambda'(u, v)$. Lastly, we define the relation $(u, v) \prec (v, w)$, to be true if and only if $(u, v)$ immediately precedes $(v, w)$ in the path $d_P$ for some $d \in D$.

Consider the following linear program:

$$\text{maximize} \sum_{d \in D} d_t - t'_{d_f} \text{ , subject to} \tag{5b.1}$$

$$t_{uv} \leq t'_{uv} \text{ for each } (u, v) \in E(G) \tag{5b.2}$$

$$t'_{uv} \leq t_{uv} + \delta \text{ for each } (u, v) \in E(G) \text{ (only for } \delta\text{-Path-DB)} \tag{5b.3}$$

$$t'_{uv} \leq t'_{vw} - 1 \text{ for each pair of edges } uv \text{ and } vw \text{ such that } uv \prec vw \tag{5b.4}$$

$$t'_{d_f} \leq d_t \text{ for each demand } d \tag{5b.5}$$

This LP has $\{t'_{uv} : (u, v) \in E(G)\}$ as its set of *unknown variables*. The variables $\{t_{uv} : (u, v) \in E(G)\} \cup \{d_t : d \in D\}$ correspond to given integers fully specified by the Path-DB instance $((G, \lambda), D)$ (and $d_f$ likewise refers to a specific edge of $G$).

**Claim 5b.7.1.** The LP is integral. Meaning: at least one optimal solution of the LP assigns integers to all of its unknown variables. Moreover, an integral solution may be recovered from a non-integral solution in polynomial time.

*Proof of claim:*

Suppose otherwise. That is, there is some non-integral solution $X$ to the LP which is strictly better than any integral solution.

Under $X$, for some edge $vw$, $t'_{vw}$ is assigned a non-integer value, say $x = y + \epsilon$ with $y \in \mathbb{N}$ and $0 < \epsilon < 1$.

Consider the assignment obtained by instead setting $t'_{vw} = y$. If this is still a valid solution to the LP, then this clearly does not worsen the objective (and cannot improve it since we assumed $X$ was optimal). Apply this update iteratively, everywhere possible, and consider the new solution $Y$ obtained. By our initial premise, $Y$ is still not an integral solution, and by construction $Y$ has the same objective value as $X$ and also would cease to be a solution if any of its non-integer variables were rounded down to the nearest integer.

We again can find some (possibly different) edge $vw$ such that $t'_{vw}$ is assigned a non-integer value under $Y$, now $y = z + \epsilon$ with $z \in \mathbb{N}$ and $0 < \epsilon < 1$.

Consider the assignment obtained by instead setting $t'_{vw} = z$. Necessarily this assignment is not a valid solution for the LP (since otherwise we already would have performed the update). Consequently, there is some constraint which is violated by the update, which necessarily has form $t'_{uv} \leq t'_{vw} - 1$, since all other types of constraints would remain satisfied if we set $t'_{uv} = z$. Moreover, $t'_{uv}$ must itself be assigned some non-integer value (strictly less than that assigned to $t'_{vw}$) under $Y$. By iteratively applying the same logic (and the fact that there are only finitely many edges) we conclude some edge must be assigned a non-integer value under $Y$ even though it could have been rounded down to the nearest integer - contradicting a central property of the assignment $Y$. We note that our construction for $Y$ may be performed in polynomial time to iteratively construct an integral solution from a non-integral one, and the claim follows. ■

Since linear programs are solvable in polynomial time [187], we may first solve the LP and then (if the solution is not already integral) apply Claim 5b.7.1 to recover an integral solution. We note here that a modification of Kahn's algorithm [188] for topological sorting may be used to compute a solution to this particular LP directly and more efficiently. A detailed proof would be quite technical and incongruous with the rest of the paper, so has been omitted.

An integral solution to this LP fully specifies a delaying $\lambda'$ satisfying the ($\delta$-)Path DB instance. Note that: $\lambda'$ is indeed a ($\delta$-)delaying of $\lambda$ (due to eqs. (5b.2) and (5b.3)); enables strict temporal paths along each path specified in $D$ (due to eq. (5b.4)); and that each of these paths reaches the destination vertex by the arrival time prescribed (due to eq. (5b.5)). Conversely, it should be clear that any delaying $\lambda'$ satisfying the ($\delta$-)Path DB instance specifies a (not necessarily optimal) solution to the LP. In fact, the LP allows us not only to *decide* ($\delta$-)Path DB, but more strongly to solve its optimization variant. □

Since trees are characterized by any pair $(u, v)$ being connected by a unique (static) path, we obtain the following corollary:

**Corollary 5b.8.** ($\delta$-)DelayBetter is in P when the underlying graph $\mathcal{G}_\downarrow$ is a (directed) tree.

Next, we are able to extend this result to "tree-like" graphs, by parameterizing by the size of the instance's feedback edge set.

**Theorem 5b.9.** On directed (reps. undirected) temporal graphs, with $|FES(\mathcal{G}_\downarrow)| = \rho$, ($\delta$-)DelayBetter is solvable in time $O(\rho! \cdot 2^{\rho \cdot |D|} \cdot \text{poly}(n))$ (resp. $O(\rho! \cdot 3^{\rho \cdot |D|} \cdot \text{poly}(n))$).

*Proof.* The proofs for directed and undirected graphs differ only in small details, and those for for $\delta$-DB and DelayBetter are identical (until we apply theorem 5b.7).

Let $E'$ be a feedback edge set of (the undirected version of) $\mathcal{G}_\downarrow$ of size $\rho$. We iterate over each of the $\rho!$ possible orderings $(e_1, e_2, ...e_\rho)$ of $E'$, and require that $t_{e_1} \leq t_{e_2} \leq ... \leq t_{e_\rho}$. (Note that if $T_{\max}$ is small and $\rho$ is large, we may prefer to iterate over all $(T_{\max})^\rho$ assignments and obtain an ordering from those.)

In any solution, each demand $d \in D$ is satisfied by a strict temporal path from $d_u$ to $d_v$ using some subset of the edges of $E'$. In the directed case, specifying this subset (together with the ordering fixed earlier) fully specifies the path from $d_u$ to $d_v$; in the undirected case, it is also necessary to specify the direction taken for each edge. The journey from one edge in the subset to the next is uniquely determined due to the fact that it can only use the edges of the spanning tree obtained by removing $E'$ from $\mathcal{G}$.

For directed graphs, this means there are at most $2^\rho$ possible paths for each demand (an edge is either chosen or not), and thus $2^{\rho \cdot |D|}$ for all demands. For undirected graphs, we get $3^\rho$ possible paths per demand (an edge $(u, v) \in E'$ is either traversed from $u$ to $v$, from $v$ to $u$, or not at all), and thus $3^{\rho \cdot |D|}$ for all demands. For each ordering of $E'$ and collection of subsets of $E'$, there is a corresponding instance of ($\delta$-)Path DB.

In total, it is sufficient to solve $\rho! \cdot 2^{\rho \cdot |D|}$ such instances of ($\delta$-)Path DB for directed graphs, and $\rho! \cdot 3^{\rho \cdot |D|}$ instances of ($\delta$-)Path DB for undirected graphs. Since ($\delta$-)Path DB is solvable in polynomial time by Theorem 5b.7, we obtain the desired result. □

## 5b.4 Hardness results

Our first two hardness results are in the restrictive setting wherein $T_{\max} = 2$ and the initial temporal assignment is the constant function $\mathbf{1}$. In this setting, the problems DelayBetter and $\delta$-DelayBetter essentially ask only whether there *exists* any $\lambda$ satisfying our passenger demands; any such $\lambda$ can be assumed without loss of generality to have lifetime 2, and could be obtained by delaying all time-edges by at most 1 - meaning our results hold for any $\delta \geq 1$.

**Theorem 5b.10.** On undirected graphs, DelayBetter (and $\delta$-DelayBetter with any $\delta \geq 1$) is NP-complete even restricted to instances where $T_{\max} = 2$, the initial temporal assignment is the constant function $\mathbf{1}$, and the $\mathcal{G}_\downarrow$ has diameter 6.

*Proof.* Our reduction is from Positive Not-All-Equal Exactly 3SAT [189], an NP-complete problem taking as input a formula $\phi$ consisting of triples of variables (which appear only positively). The formula $\phi$ is a yes-instance if there is an assignment to the variables such that every triple contains at least one true variable and at least one false variable.

We shall construct a graph $G$ which admits a temporal assignment $\lambda' : E(G) \to \{1, 2\}$ satisfying all our demands if and only if $\phi$ admits a satisfying assignment. Figure 5b.4 may be of use to the reader in following the proof. Solid (resp. dashed) edges in bold are ones which are necessarily assigned 1 (resp. 2) in any temporal assignment $\lambda'$ satisfying all demands.

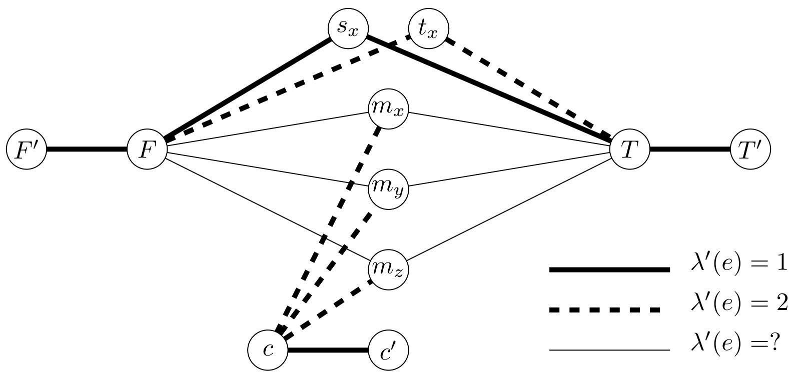


Figure 5b.4: A sketch of our construction. The vertices $s_x, t_x, m_x$ constitute the gadget for a variable $x$, and the vertices $m_x, m_y, m_z, c, c'$ constitute the gadget for a triple $c = \text{nae}(x, y, z)$.

We shall refer to the demand $(u, v, t)$ as a $t$-demand from $u$ to $v$. We begin with four special vertices $F, F', T, T'$, with 1-demands from $F'$ to $F$ and $T'$ to $T$. Then, for each variable $x$ in $\phi$, we introduce vertices $s_x, t_x, m_x$ and edges from each of these to each of $T, F$. We further introduce 1-demands from $s_x$ to each of $T, F$ (enforcing that both edges must be assigned time 1) and 2-demands from each of $T', F'$ to $t_x$ (enforcing that both of $(T, t_x), (F, t_x)$ must be assigned time 2). Lastly, we introduce 2-demands from $s_x$ to $m_x$ and from $m_x$ to $t_x$, which together with the previous constraints, guarantees that $\lambda'(m_x, T) \neq \lambda'(m_x, F)$.

Next, for each triple $c$ in $\phi$, we create vertices $c$ and $c'$ and a 1-demand between these, and connect the vertex $c$ to $m_x$ by an edge if $x$ appears in the triple $c$ and introduce a 2-demand from $c'$ to $m_x$. We also introduce 2-demands from each of $T$ and $F$ to $c$.

The intention is that assigning $\lambda'(m_x, T) = 1$ will correspond to an assignment of `true` to $x$ in $\phi$, and assigning $\lambda'(m_x, F) = 1$ will correspond to an assignment of `false` to $x$ in $\phi$. Suppose that some $\lambda'$ satisfies all demands. Then the assignment in which variable $x$ is set to `true` if $\lambda'(m_x, T) = 1$ and `false` otherwise is a satisfying assignment of $\phi$.

Suppose that $\phi$ has a satisfying assignment $X$. Consider the temporal assignment $\lambda'$ in which $\lambda'(m_x, T) = 1$ and $\lambda'(m_x, F) = 2$ if $x$ is `true` under $X$ and $\lambda'(m_x, T) = 2$ and $\lambda'(m_x, F) = 1$ otherwise (and all other values of $\lambda'$ are as specified in Figure 5b.4). Under $\lambda'$, every clause $c$ is adjacent to some pair of vertices $m_x, m_y$ such that

$x = \texttt{true}$ under $X$ and $y = \texttt{false}$ under $X$ – so the 2-demand from $T$ (resp. $F$) to $c$ can be routed through $m_x$ (resp. $m_y$). It is clear that $\lambda'$ satisfies all other demands. □

Our result for directed graphs requires a slightly different proof:

**Theorem 5b.11.** On directed graphs, DelayBetter (and $\delta$-DelayBetter with any $\delta \geq 1$) is NP-complete even restricted to instances where $T_{\max} = 2$ and $G$ has no directed cycles.

*Proof.* The reduction is again from Positive Not-All-Equal Exactly 3SAT. Given a formula $\phi$, we construct a directed graph $G$ as follows: Then, for each variable $x$ in $\phi$, we introduce six vertices $s_x, s_x^T, s_x^F, t_x, t_x^T, t_x^F$ and connect them as shown in Figure 5b.5. Further, we introduce a vertex $c$ identified with each triple $c$ in $\phi$, and create directed edges from $c$ to $s_x^T$ and from $c$ to $s_x^F$.

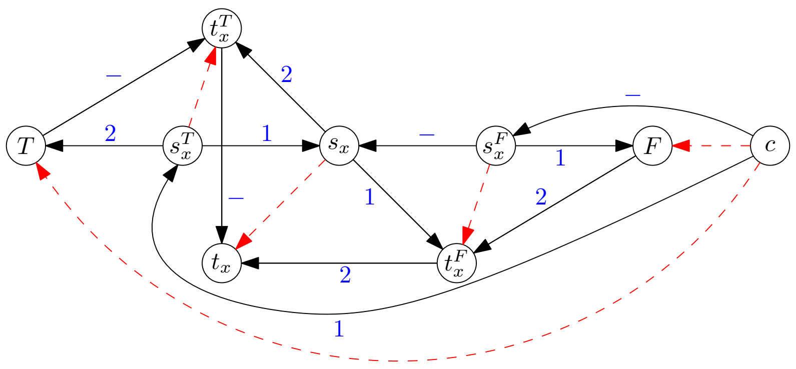


Figure 5b.5: A sketch of our construction showing NP-completeness of DelayBetter for digraphs. The vertices $s_x, s_x^T, s_x^F, t_x, t_x^T, t_x^F$ constitute the gadget for a variable $x$, and the vertex $c$ (together with its out-edges) constitutes the gadget for a triple $c \ni x$. Directed edges in $G$ are solid, whereas 2-demands are shown as dashed arrows in red. The temporal assignment shown (in blue) is one corresponding to the assignment $x$=**true** (– denotes an arbitrary choice).

We now specify the demands for our instance; for each variable $x$, we have 2-demands from $s_x$ (resp. $s_x^T$, $s_x^F$) to $t_x$ (resp. $t_x^T, s_x^F$), and for each clause $c$ we have 2-demands from $c$ to each of $T$ and $F$. (All of our demands are 2-demands, and these are shown as red dashed arrows in Figure 5b.5.) We let the constant function **1** be the initial temporal assignment for our directed graph, and this concludes the construction of our ($\delta$-)DelayBetter instance (together with specifying $\delta = 1$, if necessary).

**Claim 5b.11.1.** Let $\lambda'$ be any temporal assignment satisfying all demands in our construction. Then $\lambda'(s_x^T, T) = 2$ entails $\lambda'(s_x^F, F) = 1$, and $\lambda'(s_x^F, F) = 2$ entails $\lambda'(s_x^T, T) = 1$.

*Proof of claim:* Suppose $\lambda'(s_x^T, T) = 2$ for some $x$. Since all our demands are satisfied, we have that there must be a temporal path from $s_x^T$ to $t_x^T$ arriving at time 2. Such a path necessarily leaves at time 1 (since the two vertices are at distance 2). Consequently, $\lambda'(s_x^T, s_x) = 1$ and $\lambda'(s_x, t_x^T) = 2$. Similarly, we now must have that the 2-demand from $s_x$ to $t_x$ is routed through $t_x^F$, entailing that $\lambda'(s_x, t_x^F) = 1$

and $\lambda'(t_x^F, t_x) = 2$. Applying the same logic a third time, the 2-demand from $s_x^F$ to $t_x^F$ must be routed through $F$, and the desired claim follows. (The other direction is symmetric.) ■

Suppose that some $\lambda'$ satisfies all demands. Consider the truth assignment in which a variable $x$ is set to `true` if $\lambda'(s_x^T, T) = 2$, and `false` otherwise. Suppose for contradiction that under this truth assignment, some triple $c$ is not satisfied. Then either: (a) all variables in $c$ are `true` under our truth assignment, and leveraging Claim 5b.11.1, the vertex $c$ cannot reach the vertex $F$ by time 2; or, (b), all variables in $c$ are `false` under our truth assignment, and there $c$ cannot reach the vertex $T$ by time 2. In either case, some demand is not satisfied and we derive the desired contradiction.

Now suppose that there is some truth assignment satisfying $\phi$. Consider the temporal assignment $\lambda'$ in which:

- If $x \in c$ and $x$ is `true` (resp. `false`) under the assignment, then $\lambda'(c, s_x^T) = 1$ (resp. $\lambda'(c, s_x^F) = 1$), and
- If $x$ is `true` (resp. `false`) under the truth assignment, then $\lambda'(s_x^T, T) = 2$ (resp. $\lambda'(s_x^F, F) = 2$) and temporal assignments to other directed edges in each variable gadget being chosen consistently with the proof of Claim 5b.11.1 to satisfy demands within the variable gadget, as shown in Figure 5b.5.
- All other edges are assigned times arbitrarily.

Under $\lambda'$, $c$ has a path to $T$ (resp. $F$) through $s_x^T$ (resp. $s_x^F$) if and only if $x \in c$ is assigned `true` (resp. `false`). It should be clear that $\lambda'$ satisfies all other demands in our instance by construction, and the result follows. □

Having shown that the instance being a tree yields tractability in corollary 5b.8, we consider the case of planar graphs - a well-studied superclass of trees.

**Theorem 5b.12.** $\delta$-DelayBetter is NP-complete under any combination of the following:

- $G$ is planar and has maximum degree 10.
- Either $G$ is undirected, or $G$ is a directed acyclic graph (DAG).
- Either $T_{\max} = 19$ and $T_{\text{init}} = 1$ (with any $\delta \geq 19$), or $T_{\max} = 19$ and $\delta = 10$.

*Proof.* Our reduction is from Cubic Bipartite Planar Edge Precoloring Extension (CBP-EPE). That problem asks, given an undirected graph $G$ (which is planar, bipartite, and cubic) and a precoloring of its edges $P : E(G) \to \{R, G, B, U\}$ (indicating red, green, blue, and uncolored edges respectively) whether there is a proper edge-coloring $C : E(G) \to \{R, G, B\}$ of $G$ such that $P(e) \in \{R, G, B\} \implies C(e) = P(e)$. Let $A, B$ be an arbitrary bipartition of $V(G)$, and fix an arbitrary order on $V(G)$ (so we may refer to the $i$th neighbor of some vertex).

We shall make use of the following hardness result:

**Lemma 5b.13** (Theorem 2.3 in [190])**.** Cubic Bipartite Planar Edge Precoloring Extension is NP-complete.

**Construction** Our construction for the directed case is a specific orientation of our construction for the undirected case. Consequently, we shall describe the directed construction, which implicitly also specifies the undirected construction – but still detail explicitly, for example, that edge-gadgets can only be traversed from an $A$-gadget to a $B$-gadget (which is trivial in the directed case).

In our construction, the inclusion of a *bold time-edge* $(x, y, t)$ essentially dictates that the edge $(x, y)$ is assigned time $t$ exactly in any temporal assignment satisfying all demands. To realize this constraint, we introduce a temporal path of length and duration $t-1$ on new vertices $xy_1, \dots, xy_{t-1}$ and $x$, as shown in fig. 5b.6 and include $(xy_1, y, t)$ in our demands. Note that in the case where $t = 1$ no new vertices are created – only the demand $(x, y, 1)$.

$$\begin{tikzcd} x \arrow[r, "\mathbf{t}"] & y & := & xy_1 \arrow[r, "1"] & xy_2 \arrow[r, "2"] & \cdots \arrow[r, "t-1"] & x \arrow[r, "t"] & y & \wedge & (xy_1, y, t) \in D \end{tikzcd}$$

Figure 5b.6: Our gadget ensuring that bold time-edges are never delayed.

The reader may find the diagram in fig. 5b.7 helpful. We first describe the graph $G'$ for our instance of DelayBetter, and then the demands $D$. (For now, we let the initial temporal assignment $\lambda$ be 1 everywhere except for bold time-edges and their gadgets.)

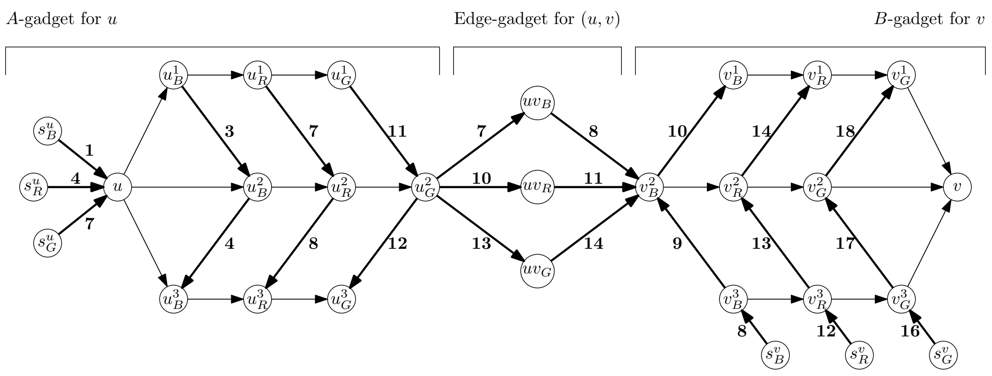


Figure 5b.7: A sketch of our reduction from Cubic Bipartite Planar Edge Precoloring Extension to DelayBetter. Only bold time-edges are labeled.

For each vertex $v \in V(G)$, we create a *vertex-gadget* consisting of a copy of $v$ and 12 other vertices $s_B^v, s_R^v, s_G^v, v_B^1, v_B^2, v_B^3, v_R^1, v_R^2, u_R^3, v_G^1, v_G^2, v_G^3$ (subscripts represent color; superscript $i$ represents the $i$th neighbor of $v$). These vertices are connected differently depending on whether $v \in A$ or $v \in B$, as shown in fig. 5b.7. In a vertex-gadget, we call *spoke edges* those edges which are not bold, and *blue* (resp. *red, green*) *layer* the vertices $v_B^i$ (resp. $v_R^i, v_G^i$).

For each edge $(u, v) \in E(G)$ with $u \in A, v \in B$, we also create three vertices $uv_B, uv_R, uv_G$. Then if $u$ is the $i$th neighbor of $v$ and $v$ is the $j$th neighbor of $u$, we introduce six bold time-edges $(u_G^i, uv_B, 7), (uv_B, v_B^j, 8), (u_G^i, uv_R, 10), (uv_R, v_B^j, 11), (u_G^i, uv_G, 13), (uv_G, v_B^j, 14)$. (In fig. 5b.7 $u$ is the second neighbor of $v$ and vice versa.) If $P(u, v)$ is precolored $G$ under $P$, then we delete two of $uv_B, uv_R$, or $uv_G$, as appropriate, leaving just one path from $u$ to $v$. In fig. 5b.7: $u$ is the second neighbor of the $v$; $v$ is the second neighbor of $u$; and $P(u,v) = U$.

We make use of three types of demands:

**Bold demands** as described earlier and shown in fig. 5b.6.

**Hermits** demands from a vertex in a vertex-gadget to another vertex in the same gadget. For each vertex $u \in A$ we have demands $(s^u_B, u^3_B, 4), (s^u_R, u^3_R, 8)$, and $(s^u_G, u^3_G, 12)$, and for each vertex $v \in B$ we have demands $(s^v_B, v, 13), (s^v_R, v, 16)$, and $(s^v_G, v, 19)$. (We say hermits have the color of the layer their source or destination lies in.)

**Travelers** demands from a vertex in an $A$-gadget to a vertex in a $B$-gadget. For each edge $(u,v)$ in the CBP-EPE instance with $u \in A$ and $v \in B$, we add a demand $(u, v, 19)$.

This concludes our construction.

## Correctness

**Claim 5b.13.1.** If the CBP-EPE instance $G, P$ is a yes-instance, then the Delay-Better instance $(G', \lambda), D$ is a yes-instance.

*Proof of claim:* We shall construct a delaying $\lambda'$ of the initial temporal assignment $\lambda$ satisfying all demands in $D$. Consider a proper edge coloring $C$ of $G$ which extends $P$.

First, we do not delay any bold time-edges – i.e., for those, $\lambda(e) = \lambda'(e)$. Note that all bold demands are immediately satisfied under any such labeling.

Let $(u,v)$ be an edge assigned color $B$ (resp. $R, G$) under $C$, with $u$ being the $i$th neighbor of $v$ and $v$ being the $j$th neighbor of $u$. We assign:

- $\lambda'(u, u^j_B) = 2$ (resp. $5, 8$)
- $\lambda'(u^j_B, u^j_R) = 3$ (resp. $6, 9$)
- $\lambda'(u^j_R, u^j_G) = 4$ (resp. $7, 10$)
- (time-edges into and out of $uv_B, uv_R, uv_G$ are all bold)
- $\lambda(v^i_B, v^i_R) = 11$ (resp. $14, 17$)
- $\lambda(v^i_R, v^i_G) = 12$ (resp. $15, 18$)
- $\lambda(v^i_G, v) = 13$ (resp. $16, 19$)

It should be clear that this labeling creates a temporal path from $u$ to $v$ for each edge $(u,v) \in G$ such that the traveler demands are satisfied (via $uv_B, uv_R$, or $uv_G$ depending on whether the edge was colored $B, R$, or $G$ under $P$).

We now show hermit demands are satisfied as well: because $P$ is a proper 3-edge-coloring of a cubic graph, every vertex is incident to exactly one edge of each color.

In $A$-gadgets, the hermit starting at $s^u_B$ (resp. $s^u_R, s^u_G$) has a temporal path to $u^i_B$ (resp. $u^i_R, u^i_G$) arriving by time 2 (resp. $6, 10$) if the edge from $u$ to its $i$th neighbor is assigned $B$ (resp. $R, G$) under $C$. The hermit can then (if $i \neq 3$) use the bold time-edges to reach $u^3_B$ (resp. $u^3_R, u^3_G$).

In $B$-gadgets, the hermit starting at $s^v_B$ (resp. $s^v_R, s^v_G$) has a temporal path to $v^i_B$ (resp. $v^i_R, v^i_G$) arriving by time 10 (resp. $14, 18$) using the bold time-edges. If the edge from $v$ to its $j$th neighbor is assigned $B$ (resp. $R, G$) under $C$, then the hermit can extend this path by using the spoke edges from $v^j_B$ (resp. $v^j_R, v^j_G$) into $v$. ■

The remainder of the proof is devoted to showing the opposite implication; that is, if DelayBetter instance $(G', \lambda), D$ is a yes-instance (i.e., there exists some delaying $\lambda'$ of $\lambda$ satisfying all demands in $D$) then the CBP-EPE instance $G, P$ is a yes-instance. For some $\lambda'$, we say that the traveler from $u$ to $v$ is *blue* (resp. *red, green*) if that traveler is routed through a vertex $uv_B$ (resp. $uv_R, uv_G$). (If several paths are possible, one may be chosen arbitrarily - though as we shall see this never happens.) No traveler has more than one color: each traveler goes through exactly one edge-gadget, from its starting $A$-gadget to its ending $B$-gadget (due to the bold time-edges enforcing the direction of the edge-gadget).

We make repeated use of the fact that, by construction, bold time-edges are never delayed. Note that if $\lambda = \mathbf{1}$ everywhere including bold gadgets, then these force their edge to be at exactly the intended time in the delaying $\lambda'$.

**Claim 5b.13.2.** Let $u \in A$. Then there is exactly one $i$ such that $\lambda'(u, u^i_B) \in [2, 4]$ (resp. $[5, 7], [8, 10]$); there is at least one $i$ such that $\lambda'(u^i_B, u^i_R) \in [6, 8]$ (resp. [9-11]); and there is at least one $i$ such that $\lambda'(u^i_R, u^i_G) \in [10, 12]$.

*Proof of claim:* The claim holds as a consequence of the hermit demands. The blue (resp. red, green) hermit must reach the blue (resp. red, green) layer using at least one (resp. two, three) spoke edge(s), arriving by time 4 (resp. 8, 12) at the latest and departing from $u$ at time 2 (resp. 5, 8) at the earliest. ■

**Claim 5b.13.3.** At most $\frac{1}{3}$ of travelers are blue and at most $\frac{2}{3}$ of travelers are red or blue.

*Proof of claim:* First, suppose over a third of travelers are blue. Then the $A$-gadget of some vertex $u$ has at least two travelers reaching different vertices of its green layer by time 6 (and, necessarily, different vertices of its red layer by time 5). This entails that at least two of the spoke edges between the blue and red layers in that gadget are at time 5 or less, which contradicts claim 5b.13.2. Similarly, if over two thirds of travelers are red or blue, then the $A$-gadget of some vertex $u$ has at least three travelers reaching three different vertices of the green layer by time 9, entailing that the three spoke edges from the red layer to the green layer are at time 9 or earlier and again contradicting claim 5b.13.2. ■

The following result is obtained through similar reasoning to that for claim 5b.13.2:

**Claim 5b.13.4.** Let $v \in B$. Then under $\lambda'$, there is some $i$ such that $v_B^i - v_R^i - v_G^i - v$ is a temporal path with departure time in $[9, 11]$ and arrival time in $[11, 13]$; there is some $i$ such that $v_R^i - v_G^i - v$ is a temporal path with departure time in $[13, 15]$ and arrival time in $[14, 16]$; and there is some $i$ such that $\lambda'(v_G^i, v) \in [17, 19]$.

*Proof of claim:* Analogously to the proof of claim 5b.13.2, we need only concern ourselves with hermits to prove this claim. The blue hermit must travel from the blue layer to $v$ as specified in the claim (since it cannot make use of any bold edges outside the blue layer in the temporal path). Similarly, the red hermit must reach $v$ by a temporal path not using any bold edges in the green layer, and the green hermit must reach $v$ using some spoke edge from the green layer in the interval $[17, 19]$. ■

**Claim 5b.13.5.** Let $v \in B$. Then at least 1 traveler arrives at the $B$-gadget of $v$ at time 8; and at least 2 travelers arrive at the $B$-gadget of $v$ at time 8 or time 11.

*Proof of claim:* First note that all travelers arriving at the $B$-gadget come from some edge-gadget and consequently arrive at a time in $\{8, 11, 14\}$. Applying claim 5b.13.4, there is some $i$ such that any traveler arriving at $v_B^i$ strictly after time 10 would be stranded there – so the traveler arriving from the $i$th neighbor of $v$ must arrive at time 8. Likewise, there is some $j$ different from $i$ such that any traveler arriving at $v_R^j$ strictly after time 14 would be stranded there – so the traveler arriving from the $j$th neighbor of $v$ must arrive at $v_R^j$ by time 14 and so at $v_B^j$ at time 8 or time 11. ■

The proof of the following is similar to that of claim 5b.13.3:

**Claim 5b.13.6.** At least $\frac{1}{3}$ of travelers are blue and at least $\frac{2}{3}$ of travelers are red or blue.

*Proof of claim:* The proof is similar to that of claim 5b.13.3. If less than a third of travelers are blue, then some $B$-gadget has all three travelers arriving strictly after time 8, contradicting claim 5b.13.5. And if less than two thirds of travelers are red or blue, then some $B$-gadget has at least two travelers arriving at time 14, again contradicting claim 5b.13.5. ■

For some $\lambda'$, we say that the traveler from $u$ to $v$ is *blue* (resp. *red, green*) if the temporal path used to route that traveler goes through a vertex $uv_B$ (resp. $uv_R, uv_G$).

**Claim 5b.13.7.** The colors of travelers in $(G', \lambda')$ fully specify a proper edge-coloring of $G$ which is consistent with the precoloring $P$.

*Proof of claim:* First, note that the precoloring is consistent with $P$ because precolored edges in $G$ have edge-gadgets consisting of only one vertex, ensuring that the traveler is assigned the appropriate color.

Next, observe that Claims 5b.13.3 and 5b.13.6 together entail that *exactly* $\frac{1}{3}$ of travelers are blue and *exactly* $\frac{1}{3}$ of travelers are red. Moreover, the proof of those claims holds locally; exactly one of the three travelers leaving any given $A$-vertex is blue (resp. red), and exactly one of the three travelers arriving at any given $B$-vertex is blue (resp. red). ■

This concludes the proof that the CBP-EPE instance $(G, P)$ is a yes-instance if the DelayBetter instance $(G, \lambda), D$ was a yes-instance.

We emphasize at this point that our construction preserves planarity and that in the directed case, the footprint contains no directed cycles. We recall that in the undirected case bold time-edges enforce that travelers can only go from an $A$-gadget to a $B$-gadget once. The maximum degree in the graph is 10 (due to vertices $u_G^i$, which are incident to 4 bold gadgets in addition to 6 normal edges). Note that the proof still holds if the initial temporal assignment $\lambda$ assigns time 2 to every non-bold edge in an $A$-gadget and time 9 to every non-bold edge in a $B$-gadget, in which case the largest delay is of 10 (delaying a time-edge from the green layer of a $B$-gadget to a $B$-vertex $v$ to be at time 19). Consequently, our proof also shows that $\delta$-DelayBetter is NP-hard for $\delta \geq 9$.

On the other hand, the proof also holds if the initial temporal assignment is instead the constant function $\mathbf{1}$: studying fig. 5b.6 it can be seen that this would still result in bold time-edges being assigned the intended time under $\lambda'$.

We have membership of NP from lemma 5b.5, and the result follows. □

## 5b.5 Discussion and open questions

We show that ($\delta$-)Path DB is in P and that ($\delta$-)DelayBetter is fpt parameterized by $|D| + \rho$, where $\rho$ is the size of smallest feedback edge set of (the undirected version of) $\mathcal{G}_{\downarrow}$. It seems likely that the techniques used in those proofs could actually solve a broader family of problems – including, for example, the natural extension of DelayBetter wherein demands specify a departure time as well as an arrival time, but also possibly problems which do not specify individual demands as part of the input. Can dependence on $|D|$ be eliminated from our fpt result? If not, then what structural parameter is sufficient to yield an fpt result without requiring $|D|$ as a parameter? A more general question for future study is: what family of temporal graph modification problems admit an fpt algorithm in the size of the feedback edge set? Separately, what is complexity of the problems parameterized by fine-grained temporal parameters (e.g., vertex interval membership width [96]), or by smaller structural parameters than $\rho$ (e.g., the feedback vertex number)? We note that for directed graphs, the size of a minimum feedback arc set (the deletion of which leaves a directed acyclic graph, or DAG) is insufficient, since we show in theorem 5b.12 that the problem is NP-complete restricted to (planar) DAGs.

Another question we leave open is: what is the complexity of DelayBetter restricted to planar inputs with $T_{\max} \in [2, 18]$? (Our proofs of Theorems 5b.10 and 5b.11 do not preserve planarity, and moreover reduce from a variant of NAE 3SAT, the restriction of which to planar instances is solvable in polynomial time [191].) Also stemming from our planar proof is the question of whether $\delta$-DelayBetter restricted to planar graphs is computationally easy or hard for values of $\delta$ below 10. Our proof was aimed at minimizing $T_{\max}$ while retaining planarity, so we expect that some easy adjustments to it might yield hardness for, e.g., $\delta = 9$, but we expect different techniques are necessary to deal with the case of $\delta = 1$ on planar graphs.

Yet another direction our investigation could be extended is to consider *non-simple* temporal graphs. Our hardness results extend immediately to this case, but our algorithms do not – in the non-simple setting, we do not expect our linear programming approach to work, and it is not even obvious whether our problems would be tractable restricted to trees.

Lastly, we observe that our results for directed and undirected versions of the problem are the same. This is particularly surprising because some of our results require substantially different proofs for each setting. An open question for future work is then: are there any natural restrictions on the input which entail instances are tractable in the directed case and computationally hard in the undirected case (or vice versa)?

# Chapter 5c

# Partial Domination

## 5c.1 Introduction

Our questions stem from Chapter 4. In that work, it was theoretically interesting and practically relevant to determine the computational complexity of Dominating Set restricted to induced subgraphs of hypercubes, but with the subtlety that we required the full hypercube (i.e. the induced supergraph) to also be in the input. It was possible, but nontrivial, to prove NP-completeness of this case. This raised the question: is there a class of graphs such that including the supergraph as a witness makes finding a dominating set strictly easier? (In the formalism introduced below: is there some $\mathcal{G}$ such that Partial Domination($\mathcal{G}$) is strictly easier than Dominating Set($\mathcal{G}_I$)?)

We note here that an adjacent but very different problem of the same name has been studied, where the input is a graph $G$ together with an integer $k$ and real number $\alpha \in (0, 1]$, and the question is whether some set of $k$ vertices dominates at least $\alpha \cdot |V|$ vertices [192, 193, 194].

## 5c.2 Prerequisites and definitions

Throughout this work, we assume P$\neq$NP (or our questions become meaningless) and, more strongly, the Exponential Time Hypothesis (ETH) - though we expect this second requirement may be unnecessary to obtain our results.

We begin with some basic definitions from graph theory. Let $G = (V, E)$ be any simple undirected graph. We define the *open neighborhood* of a vertex $v$ to be $N(v) := \{u : (u, v) \in E\}$, and its *closed neighborhood* $N[v] := N(v) \cup \{v\}$. Likewise, for any set of vertices $S$, we define $N[S] := \cup_{v \in S} N[v]$ and $N(S) := N[S] \setminus S$. A *dominating set* is a set of vertices $S \subseteq V$ such that $N[S] = V$. The *domination number* $\gamma(G)$ is the least number of vertices in any dominating set of $G$. The decision problem Dominating Set takes as input a graph $G$ and integer $k$ and asks whether $\gamma(G) \leq k$. Where $S \subseteq V$ is a set of vertices in the graph, we denote $G[S]$ the subgraph of $G$ induced by $S$. That is, $G[S]$ has $S$ as its set of vertices and as edges exactly those edges of $G$ with both endpoints incident to a vertex in $S$. We shall also make use of the following:

**Definition 5c.1** (Induced subgraph/supergraph, hereditary closure)**.** Where $T \subseteq V$ is a set of vertices in the graph, we denote $G[T]$ the subgraph of $G$ induced by $T$. That is, $G[T]$ has $T$ as its set of vertices and as edges exactly those edges of $G$ with both endpoints incident to a vertex in $T$. We say $G$ is an *induced subgraph* of $H$, and that $H$ is an *induced supergraph* of $G$, if there exists some $T$ such that $G = H[T]$.

Let $\mathcal{G}$ be a class of graphs (the object $\mathcal{G}$ is formally an infinite set of graphs). Then we denote $\mathcal{G}_I$ the family $\{H : H$ is an induced subgraph of some graph $G \in \mathcal{G}\}$. We say $\mathcal{G}_I$ is the *hereditary closure* of $\mathcal{G}$. If $\mathcal{G} = \mathcal{G}_I$, then we say the $\mathcal{G}$ is *hereditary.*

**Definition 5c.2** ($\Pi(\mathcal{G})$)**.** Let $\Pi$ be a graph problem with inputs $G, X_1, \dots, X_\ell$, with $G$ an undirected graph (possibly with $\ell = 0$, in which case $\Pi$ takes just a graph as input). We denote $\Pi(\mathcal{G})$ the restriction of $\Pi$ to the graph class $\mathcal{G}$, that is, $\Pi(\mathcal{G})$ has as instances exactly those instances of $\Pi$ satisfying $G \in \mathcal{G}$.

We introduce a new graph problem, which is subtly different from the restriction of Dominating Set, as we shall see.

Partial Domination
*Input:* Simple undirected graph $G = (V, E)$; set $T \subseteq V$; integer $k$.
*Question:* Is $(G[T], k)$ a yes-instance of Dominating Set? Equivalently, is there some set $S \subseteq T$ with $|S| \leq k$ such that $T \subseteq N[S]$?

## 5c.3 Known and immediate results

It is obvious from our definitions above that, for any class of graphs $\mathcal{G}$, if Dominating Set($\mathcal{G}$) is NP-complete, then Partial Domination($\mathcal{G}$) is NP-complete. Further, it is natural and immediate that, if Partial Domination($\mathcal{G}$) is NP-complete, then Dominating Set($\mathcal{G}_I$) is NP-complete. Polynomial solvability results propagate in the opposite direction. As implications, we may write:

$$\forall\mathcal{G} : \text{Dominating Set}(\mathcal{G}) \text{ is NP-c} \implies \text{Partial Domination}(\mathcal{G}) \text{ is NP-c} \quad (5c.1)$$

$$\forall\mathcal{G} : \text{Partial Domination}(\mathcal{G}) \text{ is NP-c} \implies \text{Dominating Set}(\mathcal{G}_I) \text{ is NP-c} \quad (5c.2)$$

$$\forall\mathcal{G} : \text{Dominating Set}(\mathcal{G}) \text{ is in P} \impliedby \text{Partial Domination}(\mathcal{G}) \text{ is in P} \quad (5c.3)$$

$$\forall\mathcal{G} : \text{Partial Domination}(\mathcal{G}) \text{ is in P} \impliedby \text{Dominating Set}(\mathcal{G}_I) \text{ is in P} \quad (5c.4)$$

On the other hand, it is not immediately clear whether the reverse implications hold. We shall exhibit classes of graphs which prove they do not.

## 5c.4 Contribution

We define, in Table 5c.1, *families of classes of graphs* $\mathfrak{A}, \dots, \mathfrak{J}$. Every graph class $\mathcal{G}$ belongs to exactly one of these families of classes depending on the computational hardness (following directly from the implications above). For example, $\mathfrak{A}_\Pi$ includes planar graphs, $\mathfrak{E}_\Pi$ includes grids, and $\mathfrak{J}_\Pi$ includes trees. (The class NP-intermediate

(NPI) contains those problems in NP which are neither in P nor NP-hard, unless P=NP.)

**Observation 5c.3.** Let $\mathcal{G}$ be a hereditary graph class (i.e. $\mathcal{G} = \mathcal{G}_I$). Then $\mathcal{G}$ necessarily belongs to one of $\mathfrak{A} \cup \mathfrak{D} \cup \mathfrak{J}$ as a consequence of eqs. (5c.1) to (5c.4).

Our principal contributions are constructive proofs that each of these families of classes is nonempty. The intention is that these preliminary findings may help to motivate further research (see Section 5c.6).

| $\mathcal{G}$ is in ... | DomSet($\mathcal{G}$) is ... | ParDom($\mathcal{G}$) is ... | DomSet($\mathcal{G}_I$) is ... | Example member |
|---|---|---|---|---|
| $\mathfrak{A}$ | NPc | NPc | NPc | planar graphs |
| $\mathfrak{B}$ | NPI | NPc | NPc | $\mathcal{B}$ (Sec. 5c.5.3) |
| $\mathfrak{C}$ | NPI | NPI | NPc | $\mathcal{C}$ (Sec. 5c.5.3) |
| $\mathfrak{D}$ | NPI | NPI | NPI | $\mathcal{D}$ (Sec. 5c.5.2) |
| $\mathfrak{E}$ | P | NPc | NPc | grids [195] |
| $\mathfrak{F}$ | P | NPI | NPc | $\mathcal{F}$ (Sec. 5c.5.3) |
| $\mathfrak{G}$ | P | NPI | NPI | $\mathcal{G}$ (Sec. 5c.5.3) |
| $\mathfrak{H}$ | P | P | NPc | $\mathcal{H}$ (Sec. 5c.5.2) |
| $\mathfrak{I}$ | P | P | NPI | $\mathcal{I}$ (Sec. 5c.5.2) |
| $\mathfrak{J}$ | P | P | P | trees |

Table 5c.1: The definition of various families of graph classes. We prove that all families are nonempty by constructing a member class for each.

## 5c.5 Results

### 5c.5.1 Tools

For ease, we shall denote $\mathcal{A}$ the class of general graphs, $\mathcal{E}$ the class of grids, and $\mathcal{J}$ the class of trees (as these belong to $\mathfrak{A}, \mathfrak{E}, \mathfrak{J}$ respectively).

**Definition 5c.4** (Leafing of a graph or graph class)**.** Where $G$ is a graph class, the *leafing* of $G$ is the graph $L(G)$ obtained by creating, for each vertex $v$ in $G$, a new vertex $l(v)$ adjacent only to $v$. (This graph may also be defined as the Corona product of $G$ and $K_1$.) Where $\mathcal{G}$ is a graph class, the leafing of $\mathcal{G}$ is the class $L(\mathcal{G}) = \{L(G) : G \in \mathcal{G}\}$.

**Definition 5c.5** ($\ell$-subdivision)**.** For $\ell \in \mathbb{N}$, the $\ell$-subdivision of a graph $G$ is a graph $G'$ where each edge $(u, v)$ is replaced by the path $\{u, uv_1, \ldots, uv_\ell, v\}$. The $\ell$-subdivision of a graph class $\mathcal{G}$ is the $\{G' : G'$ is the $\ell$-subdivision of some $G \in \mathcal{G}\}$

Note that the set of neighbors of leaves in a leafed graph is a dominating set of minimum cardinality, yielding the following:

**Lemma 5c.6.** For any graph class $\mathcal{G}$:

(i) Dominating Set$(L(\mathcal{G})) \in P$

(ii) there is a polynomial-time reduction from Partial Domination$(\mathcal{G})$ to Partial Domination$(L(\mathcal{G}))$

(iii) there is a polynomial-time reduction from Dominating Set$(\mathcal{G}_I)$ to Dominating Set$(L(\mathcal{G})_I)$.

*Proof.* (i) In forming $L(G)$ from $G$, for every new edge $(v, l(v))$, at least one of $v$ and $l(v)$ must be in any dominating set and so any minimum size dominating set of $L(G)$ has size $|G|$.

(ii) Let $(G, T, k)$ be an instance of Partial Domination$(\mathcal{G})$. Define the instance $(L(G), T, k)$ of Partial Domination$(L(\mathcal{G}))$. As $L(G)[T]$ is isomorphic to $G[T]$, $(G, T, k)$ is a yes-instance if, and only if, $(L(G), T, k)$ is a yes-instance.

(iii) Let $(G, k)$ be an instance of Dominating Set$(\mathcal{G}_I)$. Then $(G, k)$ is also an instance of Dominating Set$(L(\mathcal{G})_I$ since any induced subgraph of $G$ is an induced subgraph of $L(G)$. Trivially, $(G, k)$ is a yes-instance of Dominating Set$(\mathcal{G}_I)$ if, and only if, $(G, k)$ is a yes-instance of Dominating Set$(L(\mathcal{G})_I)$. □

**Lemma 5c.7** ([196] Theorem 4)**.** Let $G = (V, E)$ be a graph, and for each edge $e \in E$ let an integer $s(e)$ be given. Denote by $G'$ a graph obtained by a $3s(e)$-subdivision of each edge $e \in E$. Then:

A $\gamma(G') = \gamma(G) + \sum_e s(e)$

B every dominating set $D$ of $G$ can be transformed in polynomial time (in the size of $G$ and $\sum_e s(e)$) to a dominating set $D'$ of $G'$ such that $|D'| = |D| + \sum_e s(e)$

C every dominating set $D'$ of $G'$ can be transformed in polynomial time to a dominating set $D$ of $G$ such that $|D| \leq |D'| - \sum_e s(e)$

In particular, for any integer $\ell$ divisible by 3, the $\ell$-subdivision $G'$ of a graph $G$ has a dominating set of size $|E(G)| \cdot \frac{l}{3} + k$ if, and only if, $G$ has a dominating set of size $k$.

We shall make use of the following conjecture, as is common in computational complexity:

**Definition 5c.8** (Exponential Time Hypothesis (ETH))**.** 3-SAT cannot be solved in $2^{o(n)}$ time.

The ETH is strictly stronger than P=NP. Note that our line of investigation only makes sense assuming P$\neq$ NP - though ETH is not strictly necessary, and we conjecture that our results could be obtained without relying on it. The class QP contains all decision problems which can solved in *quasi-polynomial time* $O(n^{\text{polylog}(n)})$.

**Lemma 5c.9.** The class QP ∩ NP-complete is empty unless the ETH fails.

*Proof.* If there is such a $\Pi$, then reducing a 3-SAT instance $\phi$ to the problem is doable in polynomial time (by NP-completeness) and then solving the obtained instance of $\Pi$ is doable in time $|I|^{\text{polylog}(|I|)}$ (by containment in QP). Then solving $\phi$ "through" $|I|$ takes time at most $(|\phi|^a)^{\log^b(|\phi|^a)} \in |\phi|^{\text{polylog}|\phi|}$ and so 3-SAT is in QP, so ETH fails. □

**Lemma 5c.10** ([197])**.** There is a $O(3^{\boldsymbol{tw}}n)$ parameterized algorithm solving Dominating Set, where $\boldsymbol{tw}$ is the treewidth of the input graph.

**Lemma 5c.11.** If $G$ is a graph with $k \geq 4$ vertices of degree at least 3, then $G$ has treewidth at most $k-1$. In particular, for any $\ell \in \mathbb{N}$, (any induced subgraph of) the $\ell$-subdivision of a graph on $k \geq 4$ vertices has treewidth at most $k-1$.

*Proof.* Let $G$ be a connected graph (otherwise apply this proof to each component separately) and let $S(G)$ be the set of $k$ vertices of degree at least 3 in $G$. Observe that $G - S(G)$ is a disjoint union of paths. We now describe how to construct a tree decomposition of the desired width – the reader may find the illustration in Figure 5c.1 helpful. First include $S(G)$ as a bag in the decomposition. Then, for each such path $P = (p_1, p_2, \ldots, p_\ell)$:

- if both endpoints of $P$ are adjacent to vertices $u$ and $v$ in $S(G)$ (possibly with $u = v$), then create bags $\{u, p_1, v, p_\ell\}, \{p_1, p_2, p_{\ell-1}, p_\ell\}, \ldots$, moving outward from $S(G)$ towards the midpoint of $P$,
- if only one endpoint of $P$ (without loss of generality $p_1$) is adjacent to some vertex $u$ in $S(G)$, then create bags $\{u, p_1\}, \{p_1, p_2\}, \ldots, \{p_{\ell-1}, p_\ell\}$, moving outward from $S(G)$ towards the the other end of $P$,

The largest bag in the described decomposition has size $\min(S(G), 4)$. Applying the fact that the treewidth of an induced subgraph of some graph $G$ is at most that of $G$, the result follows. □

### 5c.5.2 $\mathfrak{D}, \mathfrak{H}, \mathfrak{I}$ are nonempty.

Let $\mathcal{D}$ be the hereditary closure of the set of graphs obtained by subdividing $n$-vertex graphs $\lceil 2^{\sqrt{n}} \rceil_3$ times, where $\lceil x \rceil_3$ denotes the least multiple of three which is at least $x$.

$$\mathcal{D} := \left\{ G \text{ subdivided } \lceil 2^{\sqrt{|V(G)|}} \rceil_3 \text{ times } : G \in \mathcal{A} \right\}_I$$

**Lemma 5c.12.** Let $D \in \mathcal{D}$. Then the treewidth of $D$ is in $O(\log^2(|V(D)|)$.

*Proof.* Let $S(D) \subseteq V(D)$ be the set of supercubic vertices in the graph, (i.e., those with three or more neighbors) and denote $n_S = |S(D)|, n_V = |V(D)|$. Then one of the following holds:

- $n_V \geq 2^{\sqrt{n_S}}$, and consequently $n_S \leq \log^2 n_V$ (applying Lemma 5c.11, we have $\text{tw}(D) \leq n_S$), or

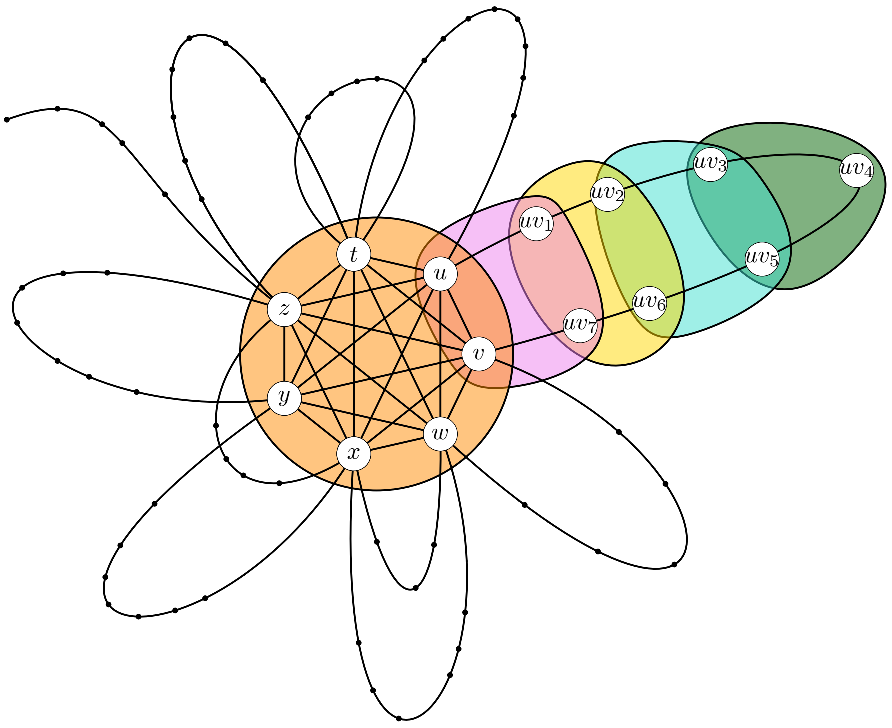


Figure 5c.1: Illustration of a tree decomposition of a small graph $G$ with $|S(G)| = 7$.

- every connected component of $D$ is an isolated vertex, a subdivided path, or a subdivided star graph, and $\text{tw}(D) = 1$.

In either case, we obtain the desired bound on $\text{tw}(D)$, and the result follows. □

**Theorem 5c.13.** $\mathcal{D} \in \mathfrak{D}$.

*Proof.* First observe that $\mathcal{D}$ is hereditary. Applying observation 5c.3, we need only show that Dominating Set($\mathcal{D}$) is NP-intermediate to show $\mathcal{D} \in \mathfrak{D}$.

**Claim 5c.13.1.** There can be no polynomial-time algorithm for Dominating Set($\mathcal{D}$) unless the ETH fails.

*Proof of claim:* Suppose there is an algorithm `A` solving Dominating Set($\mathcal{D}$) in polynomial time. That is, there is a constant $c$ so that for any $G_D \in \mathcal{D}$ and $k_D \in \mathbb{N}$, $\mathtt{A}(G_D, k_d)$ runs in time $O(|V(D)|^c)$ and returns `true` if and only if $\gamma(G) \leq k_D$.

Now let $(G, k)$ be an instance of Dominating Set($\mathcal{A}$) with $n$ vertices and $m$ edges. Let $G_D$ be the graph obtained by subdividing $G$ $\lceil 2^{\sqrt{n}} \rceil_3$ times, and note that $G_D \in \mathcal{D}$. Also, let $k_D = m \cdot \frac{\lceil 2^{\sqrt{n}} \rceil_3}{3} + k$, so that (applying lemma 5c.7) $G_D$ admits a dominating set of size $k_D$ if and only if $G$ admits a dominating set of size $k$. Applying the construction of $G_D$ (which will have size $n_D = m\lceil 2^{\sqrt{n}} \rceil_3 \leq n^2 2^{\sqrt{n}} = 2^{2\log(n)\sqrt{n}}$) and then the algorithm `A` takes time $O({n_D}^c) \leq O(2^{2c\log(n)\sqrt{n}})$ - yielding the desired contradiction: a subexponential algorithm for Dominating Set. ■

**Claim 5c.13.2.** Dominating Set($\mathcal{D}$) is in QP.

*Proof of claim:* We have from Lemma 5c.12 that the treewidth of any graph $D$ in $\mathcal{D}$ is in $O(\log^2(|V(D)|)$. Combining this result with the Dominating Set algorithm from Lemma 5c.10 which has runtime $O(3^{\text{tw}} n)$, we obtain an algorithm with runtime $O(3^{\log^2(n)} n)$. ■

From claim 5c.13.1 we have that DOMINATING SET($\mathcal{D}$) is not solvable in polynomial time, and by applying claim 5c.13.2 and lemma 5c.9, it follows that DOMINATING SET($\mathcal{D}$) is not NP-complete. Consequently, DOMINATING SET($\mathcal{D}$) is NP-intermediate. □

We now apply this result to populate $\mathfrak{H}$ and $\mathfrak{I}$. First, let $\mathcal{H} := \left\{G \oplus \overline{K_{2^{|V(G)|}}} : G \in \mathcal{A}\right\}$. That is, $\mathcal{H}$ is the family of graphs containing, for each graph $G$ on $n$ vertices (recall $\mathcal{A}$ is the class of general graphs), the disjoint union of $G$ and $2^n$ isolated vertices.

**Theorem 5c.14.** *$\mathcal{H} \in \mathfrak{H}$.*

*Proof.* First observe that $\mathcal{H}_I = \mathcal{A}$, immediately yielding that DOMINATING SET($\mathcal{H}_I$) is NP-complete. Also note that, given any instance of PARTIAL DOMINATION($\mathcal{H}$), we may discard all singleton vertices from the instance (reducing the target $k$ as appropriate if they are in the set $T$) to obtain a "kernel" instance of size logarithmic in the size of the original graph. We then may apply a brute-force algorithm to solving the kernel instance in time polynomial in the size of the original input (and exponential in the size of the kernel). Applying implication eq. (5c.3) the result also holds for DOMINATING SET($\mathcal{H}$). □

We now combine both ideas above: let $\mathcal{I} := \left\{D \oplus \overline{K_{2^{|V(G)|}}} : D \in \mathcal{D}\right\}$. That is, $\mathcal{I}$ is the family of graphs containing, for each graph $G$ on $n$ vertices from $\mathcal{D}$, the disjoint union of $G$ and $2^n$ isolated vertices.

**Theorem 5c.15.** *$\mathcal{I} \in \mathfrak{I}$.*

*Proof.* First observe that $\mathcal{H}_I = \mathcal{D}$, so DOMINATING SET($\mathcal{H}_I$) is NP-intermediate (applying Theorem 5c.13). Applying the same logic as in our proof of Theorem 5c.14, any instance of PARTIAL DOMINATION($\mathcal{H}$) may be reduced to a "kernel" instance of size logarithmic in that of the original, and subsequently we may apply a brute-force algorithm to solving the kernel instance in time polynomial in the size of the original input. Again applying implication eq. (5c.3) the result also holds for DOMINATING SET($\mathcal{I}$). □

### 5c.5.3 $\mathfrak{B}, \mathfrak{C}, \mathfrak{F}, \mathfrak{G}$ are nonempty.

Using the results above, it becomes relatively straightforward to prove the following:

- $\mathfrak{B} \ni \mathcal{B} := \mathcal{E} \cup \mathcal{D}$,
- $\mathfrak{C} \ni \mathcal{D} \cup \mathcal{H}$,
- $\mathfrak{F} \ni \mathcal{F} := L(\mathcal{C})$ (applying Lemma 5c.6), and
- $\mathfrak{G} \ni \mathcal{G} := L(\mathcal{D})$ (again applying Lemma 5c.6)

## 5c.6 Further questions

Without assuming P$\neq$NP our problems become vacuous; nonetheless, it seems likely that many of our results could be proven without assuming ETH.

**Question 5c.1.** Can we remove our reliance on ETH?

Also, we note that the classes we construct in this chapter are not especially natural, and expect that these have not been of previous interest.

**Question 5c.2.** Which (if any) *natural* classes belong to $\mathfrak{B}, \mathfrak{C}, \mathfrak{D}, \mathfrak{F}, \mathfrak{G}, \mathfrak{H}, \mathfrak{I}$?

One particularly interesting candidate is the family of hypercubes $\mathcal{Q}$.

**Question 5c.3.** Is there a polynomial-time algorithm for Dominating Set($\mathcal{Q}$)?

It is unlikely there is a practical polynomial-time algorithm for Dominating Set($\mathcal{Q}$); even $\gamma(Q_{10})$ is unkown. See `https://oeis.org/A000983` and the discussion therein. On the other hand, it is provable (by leveraging Mahaney's Theorem [198]) that Dominating Set($\mathcal{Q}$) is not NP-c unless P=NP. If the answer to Question 5c.3 is negative, this would provide a first natural class belonging to $\mathfrak{B}$ (if the answer is positive, then $\mathcal{Q} \in \mathfrak{E}$, the same as grids).

## Chapter 5d

# A nifty Constraint Satisfaction Problem

## 5d.1 Introduction

We are consider restrictions of the Constraint Satisfaction Problem (CSP) 1-IN-3.

> 1-IN-3
>
> *Input:* a formula $\phi$ consisting of a sequence of triplets of boolean variables $X = \{x_i | 1 \leq i \leq n\}$.
>
> *Question:* is there an assignment to $X$ such that for each triplet $t_j \in \phi$, exactly one variable in $t_j$ is assigned `True`?

One way of expressing constraints on $\phi$ is to consider the hypergraph it directly describes, as illustrated in Fig. 5d.1. Note that such a hypergraph is necessarily 3-uniform: that is, each hyperedge has cardinality 3.

$$\phi = (x, y, z) \wedge (u, v, z) \wedge (u, y, w)$$

$$X = \{u, v, w, x, y, z\}$$

$$\mathcal{E} = \{(x, y, z), (u, v, z), (u, y, w)\}$$

$$H_\phi = (X, \mathcal{E})$$

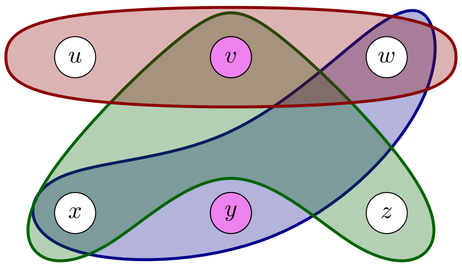


Figure 5d.1: A 1-IN-3 instance $\phi$ and the corresponding hypergraph $H_\phi$. The vertices $y$ and $v$ are an independent vertex cover of the hypergraph $H_\phi$, and setting $y = v =$`true` and all other variables `false` is a satisfying assignment for $\phi$.

Note that an equivalent formulation of the problem itself also follows: 1-IN-3 with input $\phi$ asks whether there is an independent vertex cover (IVC) in the 3-uniform hypergraph $H_\phi$ (i.e., a set $S$ of vertices in the hypergraph such that each hyperedge intersects $S$ in exactly one vertex), or equivalently whether there is a vertex cover of cardinality exactly one third the number of hyperedges. VERTEX COVER in (hyper)graphs is a classic combinatorial problem which has been the subject of extensive study for decades [10, 199, 200]. Closely related is the notion of a *perfect* *matching* in a hypergraph – a disjoint set of hyperedges the union of which is equal to the set of all vertices in the hypergraphs. This problem, too, has a rich literature (see, e.g., [201, 202]). 3-Dimensional Matching, which is one of Karp's 21 NP-complete problems [10], is a special case:

3 Dimensional Matching (3DM)
*Input:* sets $X, Y, Z$, triples $T \subseteq X \times Y \times Z$, and integer $k$.
*Question:* does there exist a subset $S \subseteq T$ of triples of size $k$ such that the triples in $S$ are disjoint?

We shall make frequent use of the connection between 1-in-3 and Independent Vertex Cover for the remainder of this chapter, for example using the statements "(the vertex) $v$ is (not) in the IVC" and "(the Boolean variable) $v$ is assigned `true` (resp. false) in the satisfying assignment" interchangeably.

It is known that 1-in-3 remains NP-complete when restricted to *cubic* instances:

Cubic 1-in-3
*Input:* a 1-in-3 formula $\phi$ in which every variable appears exactly three times.
*Question:* is there a satisfying assignment for $\phi$?

**Lemma 5d.1** ([203] Theorem 29)**.** Cubic 1-in-3 is NP-complete.

Note that [203] uses the language of XSAT to describe the problem, stating that it remains NP-complete restricted to $k$-CNF$^l_+$ (i.e., CNF formulas where variables appear exactly $l$ times and only positively, and clauses all have size $k$) for any $k, l \geq 3$.

**Tricolor 1-in-3** We say that $\phi$ is tricolor if the chromatic index of $H_\phi$ is three – so each constraint in $\phi$ can be assigned a color in {red, blue, green} so that no two overlapping constraints also share a color. When $\phi$ is tricolor, each color class of hyperedges is a matching, so we can draw hyperedges as triangles of the appropriate color without the risk of ambiguity (as in the center of Figure 5d.2).

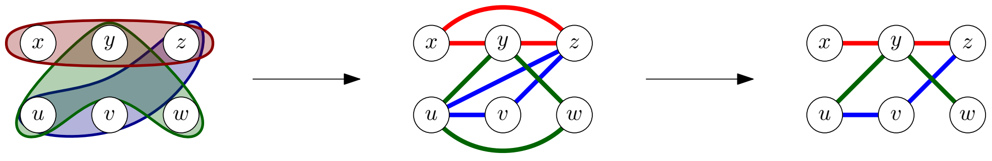


Figure 5d.2: Representing hyperedges in a tricolor instance of 1-in-3 as an edge-colored graph.

We can also optionally omit one of the three edges of the triangle when this makes the drawing simpler; if $(x, y)$ and $(y, z)$ are green edges, then we know $(u, v, w)$ is a constraint (as on the right of Figure 5d.2)).

**Forbidding triangles** A *triangle* in a hypergraph is a set of three hyperedges which have pairwise non-empty intersections (see e.g., [204]). In a tricolor hypergraph,

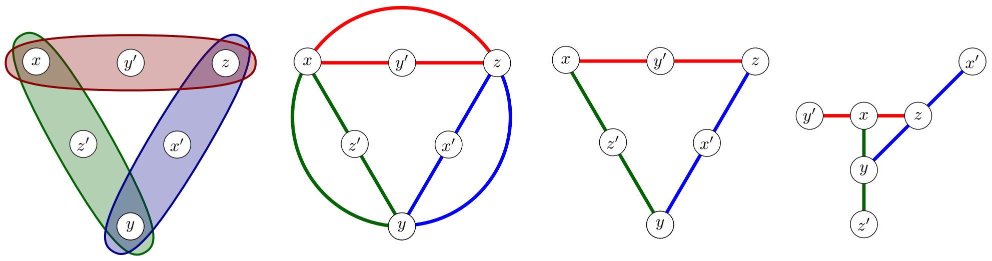


Figure 5d.3: Four different drawings of a triangle in a simple tricolor hypergraph.

a triangle necessarily consists of one hyperedge of each color (see Figure 5d.3 and Figure 5d.14). A *triangle-free* hypergraph is one in which no subset of hyperedges forms a triangle. A *simple* hypergraph is one in which no two constraints have overlap in two or more variables. Combining these, we obtain that a triangle-free simple hypergraph is a hypergraph of girth at least 4. That is, its smallest (Berge-)cycle has length at least 4 [205].

**Our problems** are then the following:

Tricolor Cubic 1-in-3
*Input:* a 1-in-3 formula $\phi$ in which every variable appears exactly three times **and** such that $H_\phi$ admits a 3-hyperedge coloring (which may be assumed to be provided as part of the input).
*Question:* is there a satisfying assignment for $\phi$?

Triangle-free Tricolor Cubic Simple 1-in-3
*Input:* a 1-in-3 formula $\phi$ in which every variable appears exactly three times **and** such that $H_\phi$ admits a 3-hyperedge coloring (provided), is triangle-free, and is simple.
*Question:* is there a satisfying assignment for $\phi$?

## 5d.2 Tricolor Cubic 1-in-3 is NP-complete

**Theorem 5d.2.** Tricolor Cubic 1-in-3 is NP-complete.

*Proof.* Membership of NP follows straightforwardly being a subproblem of 1-in-3. Our reduction is from Cubic 1-in-3, which is known to be NP-complete by Lemma 5d.1. Denote $X_{123} = \{x_1, x_2, x_3 : x \in X\}$ and let $\phi_{123}$ be the formula $\phi$ in which the $i$th appearance of each variable $x \in X$ is replaced by $x_i$. For example we may have: $\phi = (x, y, z) \wedge (x, y, w) \dots$ and $\phi_{123} = (x_1, y_1, z_1) \wedge (x_2, y_2, w_1) \dots$. Note that each variable in $X_{123}$ appears exactly once in $\phi_{123}$. We say that all the constraints of $\phi_{123}$ are colored blue. We now seek to enforce that $x_1 = x_2 = x_3$.

**Enforcing equality** Let $x \in X$ be a variable from $\phi$. Introduce new *auxiliary variables* $A^x = \{a_i^x : 1 \le i \le 6\}$ and *dummy variables* $D^x = \{d_1^x, d_2^x, d_3^x\}$. Then $\psi(x)$ consists of the following constraints:

**Green constraints** $(x_1, a_1^x, a_2^x), (a_4^x, x_2, a_3^x), (a_5^x, a_6^x, x_3)$

**Red constraints** $(x_1, a_4^x, a_5^x), (a_1^x, x_2, a_6^x), (a_2^x, a_3^x, x_3)$

**Blue constraints** $(a_1^x, a_3^x, d_1^x), (a_2^x, a_5^x, d_2^x), (a_3^x, a_6^x, d_3^x)$

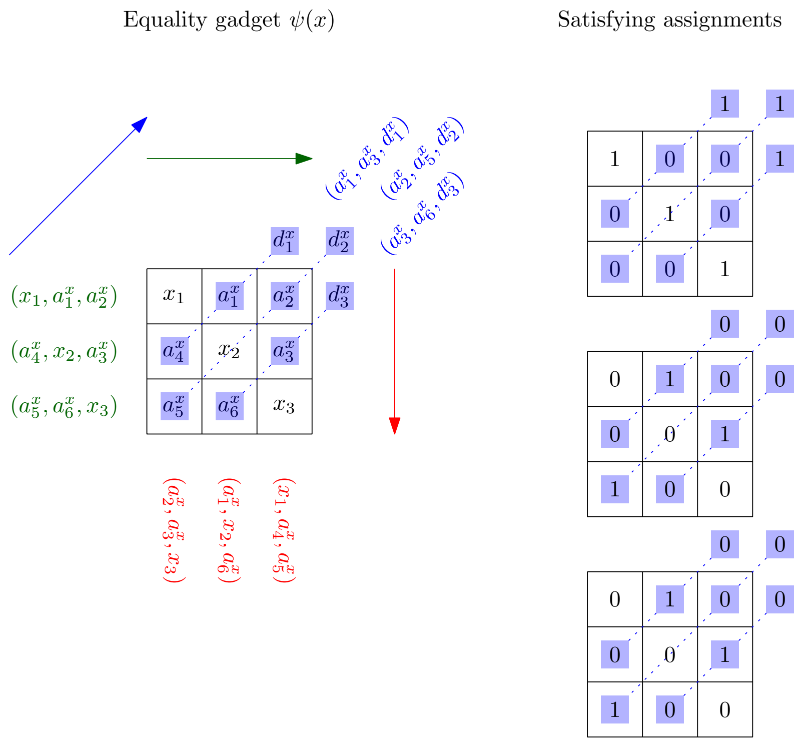


Figure 5d.4: The equality gadget $\psi(x)$ and its satisfying assignments.

Note that if $\psi(x)$ is satisfied (with exactly one True variable per constraint) then $x_1 = x_2 = x_3 = d_1 = d_2 = d_3$, and moreover $\psi(x)$ has a satisfying assignment with $x_1 = 1$ and two satisfying assignments with $x_1 = 0$. We show the gadget and all satisfying assignments in fig. 5d.4.

Denote $\psi(X)$ the conjunction $\bigwedge_{x \in X} \psi(x)$. Then $\phi_{123} \wedge \psi(X)$ is a yes-instance of 1-IN-3 iff $\phi$ is a yes-instance of CUBIC 1-IN-3. Note also that each variable appears in a blue constraint, a red constraint, and a green constraint, except for dummy variables, which appear only in blue constraints. We now seek to tie up this loose end by introducing a gadget which will guarantee that each dummy variable additionally appears in a green constraint and a red constraint.

**Balancing colors** Let $c$ be a constraint with $c = (x, y, z)$ and $i, j, k \in \{1, 2, 3\}$ be integers such that $x$'s $i$th appearance is in $c$, $y$'s $j$th appearance is in $c$, and $z$'s $k$th appearance is in $c$. Introduce new variables $B^c = \{b_i^c : 1 \le i \le 6\}$. Then $\eta(c)$ consists of the following constraints:

**Green constraints** $(d_i^x, b_1^c, b_2^c), (b_4^c, d_j^y, b_3^c), (b_5^c, b_6^c, d_k^z)$

**Red constraints** $(d_i^x, b_4^c, b_5^c), (d_i^x, b_4^c, b_5^c), (b_2^c, b_3^c, d_k^z)$

**Blue constraints** $(b_1^c, b_2^c, b_3^c), (b_4^c, b_5^c, b_6^c)$

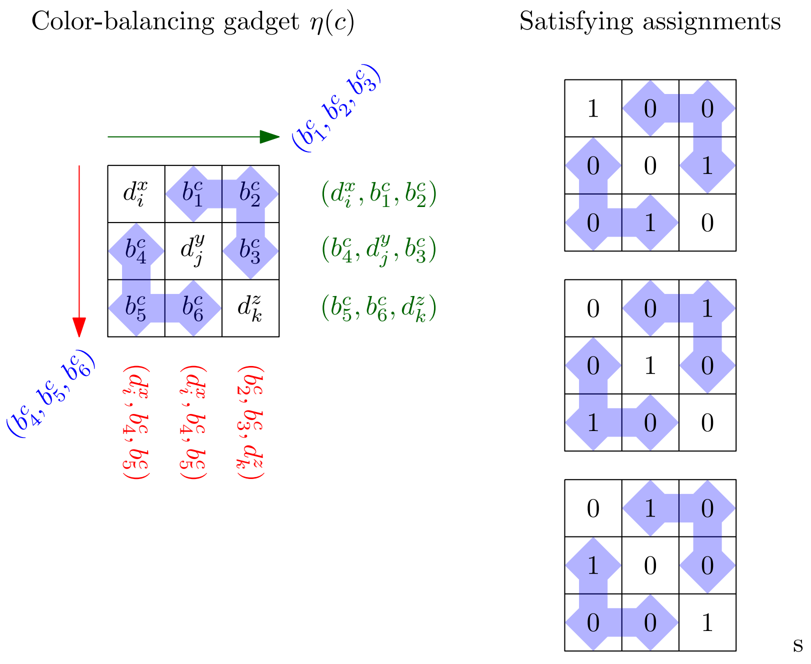


Figure 5d.5: The color-balancing gadget $\eta(x)$ and its satisfying assignments.

Note that any assignment satisfying $\phi_{123} \wedge \psi(X)$ (and in particular the constraint corresponding to $c$ in $\phi_{123}$), necessarily also satisfies $(d_i^x, d_j^y, d_k^z)$ (i.e. exactly one of those 3 is set to True, even though that constraint is not explicit). On the right side of fig. 5d.5, we show a satisfying assignment to $\eta(c)$ corresponding to each such assignment to $\{d_i^x, d_j^y, d_k^z\}$.

**Tying things together** Denote $\eta(\phi)$ the conjunction $\bigwedge_{c \in \phi} \eta(c)$. Then $\phi_{123} \wedge \psi(X) \wedge \eta(\phi)$:

- Is clearly Tricolor and Cubic by construction: each variable appears in exactly one red constraint , exactly one green constraint, and exactly one blue constraint.
- Admits a satisfying assignment if and only if $\phi$ admits a satisfying assignment.

Consequently, Cubic 1-in-3 $\leq_{\text{poly}}$ Tricolor Cubic 1-in-3 and so Tricolor Cubic 1-in-3 is NP-complete. □

## 5d.3 Triangle-free Tricolor Cubic Simple 1-in-3 is NP-complete

We shall require the following lemma:

**Lemma 5d.3.** Let $H$ by the 9-vertex hypergraph shown in the top-left of Figure 5d.6, and let $H'$ be a hypergraph obtained from $H$ by the removal of exactly one hyperedge. Then the set of IVCs of $H$ is identical to the set of IVCs of $H'$.

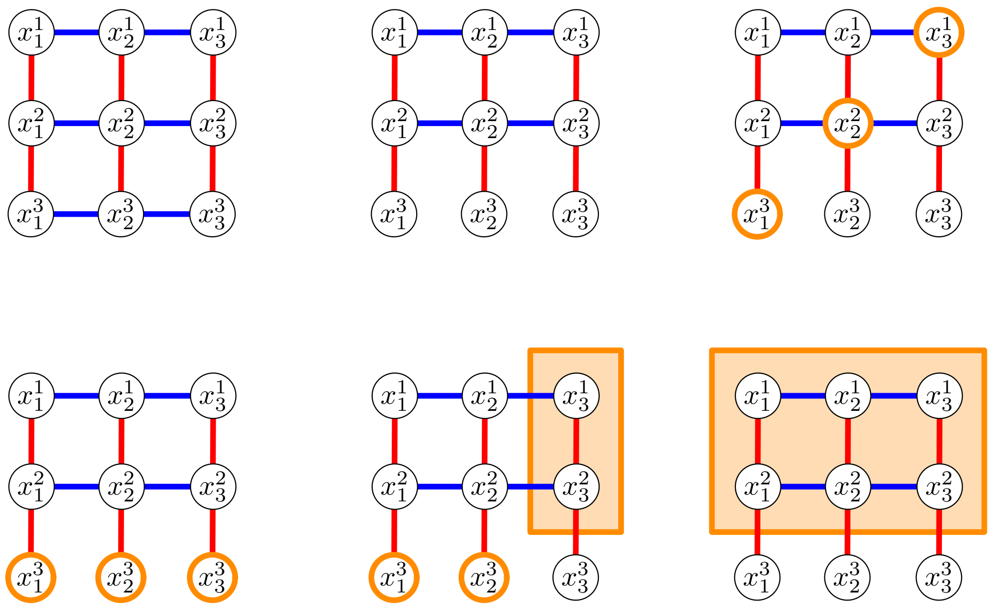


Figure 5d.6: Top left: the 9-vertex hypergraph $H$. Top center: the hypergraph $H'$ (the removal of any other hyperedge results in an isomorphic outcome). Top right: an IVC of $H'$. Bottom, left to right: illustrations of cases (a-c) of our proof.

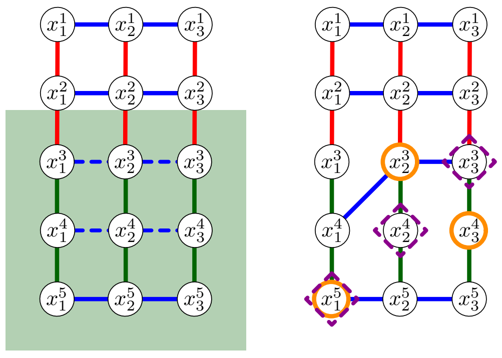


Figure 5d.7: Using Lemma 5d.3. The deletion of either or both of the dashed blue hyperedges in the hypergraph on the left does not change the set of IVCs. (We highlight such "simulated grids" in dark green in later figures as well.) Moreover, any IVC of the hypergraph on the right (obtained from that on the left by adding the hyperedge $(x_1^4, x_2^3, x_3^3)$) is a superset of **either** $\{x_1^5, x_3^4, x_2^3\}$ (shown in bold orange circles), **or** $\{x_1^5, x_3^4, x_2^3\}$ (shown in dashed purple diamonds).

*Proof.* Let $e = (x_1^3, x_2^3, x_3^3)$ be the deleted hyperedge. Clearly, any IVC of $H$ is also an IVC of $H'$. It remains to show that any IVC $I$ of $H'$ is also an IVC of $H$ (for example). Suppose for contradiction that it is not (i.e., $|S \cap I| \neq 1$). We must consider the following cases:

(a) If $|e \cap I| = 3$, then in $H'$ the hyperedge $(x_1^1, x_2^1, x_3^1)$ cannot be covered.

(b) If $|e \cap I| = 2$, then in $H'$ the hyperedge $(x_1^1, x_2^1, x_3^1)$ can be covered only if $x_3^1 \in I$ – but then the hyperedge $(x_1^2, x_2^2, x_3^2)$ cannot be covered.

(c) If $|e \cap I| = 0$, then in $H'$ the two hyperedge $(x_1^1, x_2^1, x_3^1)$ and $(x_1^2, x_2^2, x_3^2)$ must together contain three vertices.

In any case, we derive a contradiction, and the result follows. $\square$

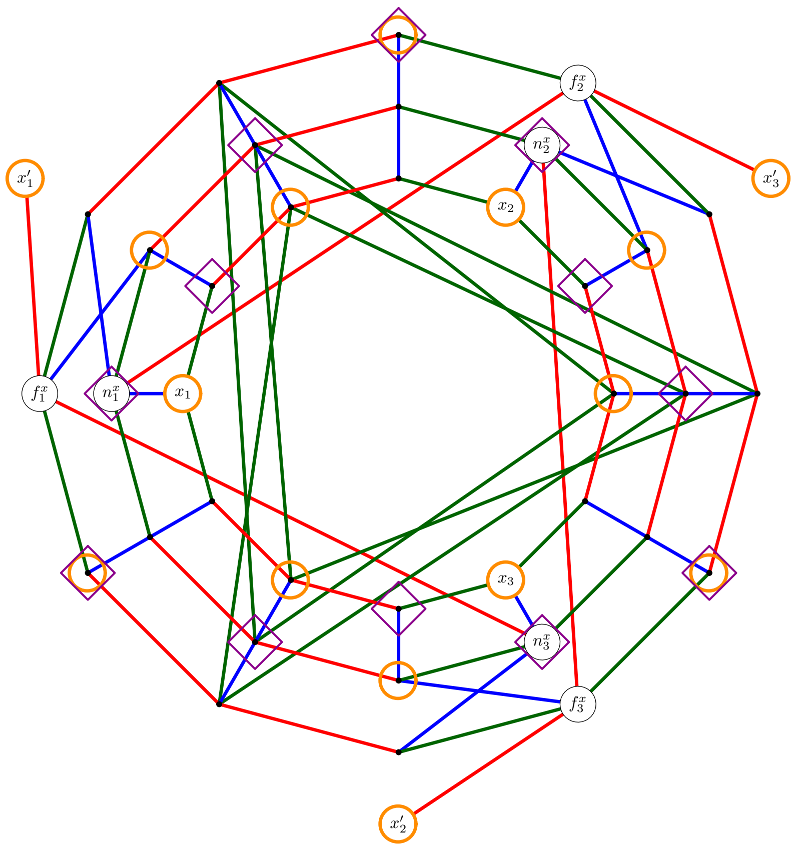


Figure 5d.8: The equality gadget $\psi(x)$. The two possible IVCs for thse gadget are shown as highlighted with orange circles and purple diamonds, respectively.

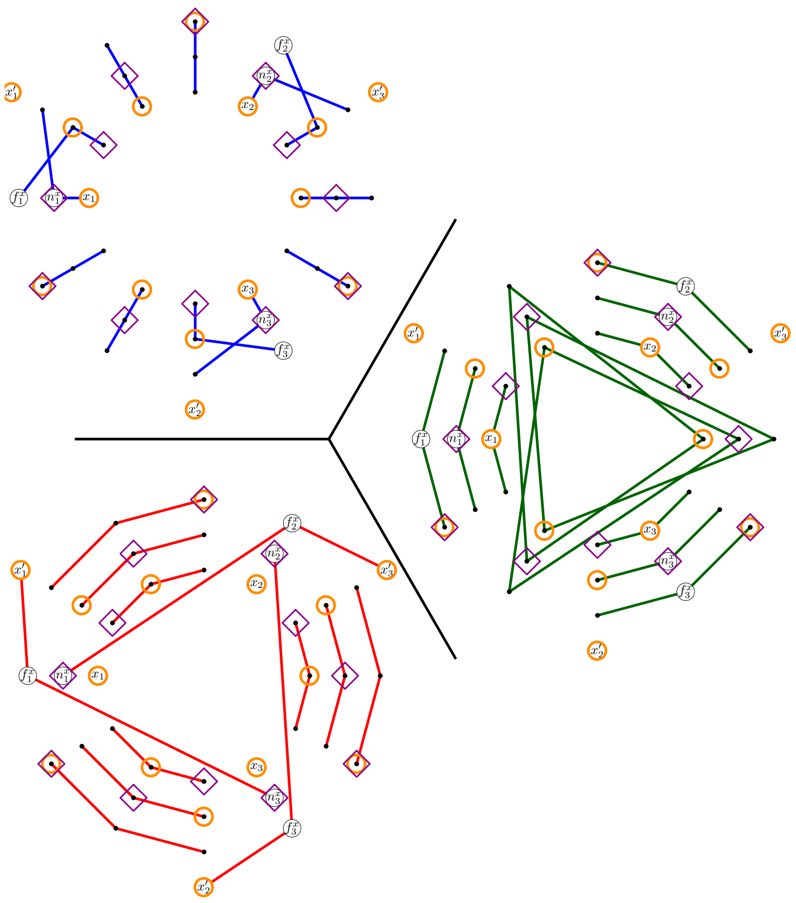


Figure 5d.9: The equality gadget $\psi(x)$ by hyperedge color: blue hyperedges are shown top left, green hyperedges center right, and red hyperedges bottom left. The two possible IVCs for the gadget are shown as highlighted with orange circles and purple diamonds, respectively.

We are now ready to prove our main result:

**Theorem 5d.4.** Triangle-free Tricolor Cubic Simple 1-in-3 is NP-complete.

*Proof.* The reduction is from Tricolor Cubic 1-in-3, which we showed was NP-complete in Theorem 5d.2. Our proof here is structured similarly to that one (although each step is somewhat more involved). First, we describe an *equality gadget* which can be used to "duplicate" a variable $x$ into three copies $x_1, x_2, x_3$, which we then show must all be assigned the same value in any satisfying assignment. Second, we provide a color-balancing gadget. Lastly, we give some concluding remarks and justify that the obtained instance is indeed cubic, simple, tricolor, and triangle-free.

Let $\phi$ be an instance of Tricolor Cubic 1-in-3. We do not actually use the fact that $\phi$ is Tricolor and Cubic, but do make use of the (strictly weaker) property that there is a subset of the constraints in $\phi$ which contains each variable exactly once (that is, $H_\phi$ admits a perfect matching). Denote this matching $M_\phi$.

Denote $X_{123} = \{x_1, x_2, x_3 : x \in X\}$ and let $\phi_{123}$ be the formula $\phi$ in which the $i$th appearance of each variable $x \in X$ is replaced by $x_i$. Note that each variable in $X_{123}$ appears exactly once in $\phi_{123}$. We say that all the constraints of $\phi_{123}$ are colored red. We now seek to enforce that $x_1 = x_2 = x_3$.

**Enforcing Equality** Let $x \in X$ be a variable from $\phi$. Our *equality gadget* $\psi(x)$ is shown in Figures 5d.8 to 5d.10. The reader is encouraged to make use of the different figures to verify different aspects of the construction, as needed – for now, Figure 5d.8 should suffice. The equality gadget has 36 new variables specific to $x$ in addition to the variables $x_1, x_2, x_3$ already introduced in $\phi_{123}$.

It is easy to verify (especially one color at a time, making use of the different figures):

- That the gadget is triangle-free, tricolor, and simple (and almost cubic; only $x_1, x_2, x_3, x_1'$, $x_2', x_3'$ are **not** incident to exactly three constraints in the gadget - recall that $x_1, x_2, x_3$ are incident to constraints in $\phi_{123}$).
- That the orange assignment (where variables are True iff they are marked with a bold orange circle) and the purple assignment (where variables are True iff they are marked with a bold purple diamond) each satisfy the constraints of the gadget (i.e., are IVCs of the hypergraph).

We now prove a stronger claim:

**Claim 5d.4.1.** Any IVC for the equality gadget contains the orange vertices, or contains the the purple vertices. (And consequently contains no other vertices from the gadget.)

*Proof of claim:* Let $I$ be an IVC for the gadget. We consider two cases: either $x_1 \in I$ or $x_1 \notin I$. As we shall see, the former must result in the orange assignment and the latter must result in the purple assignment.

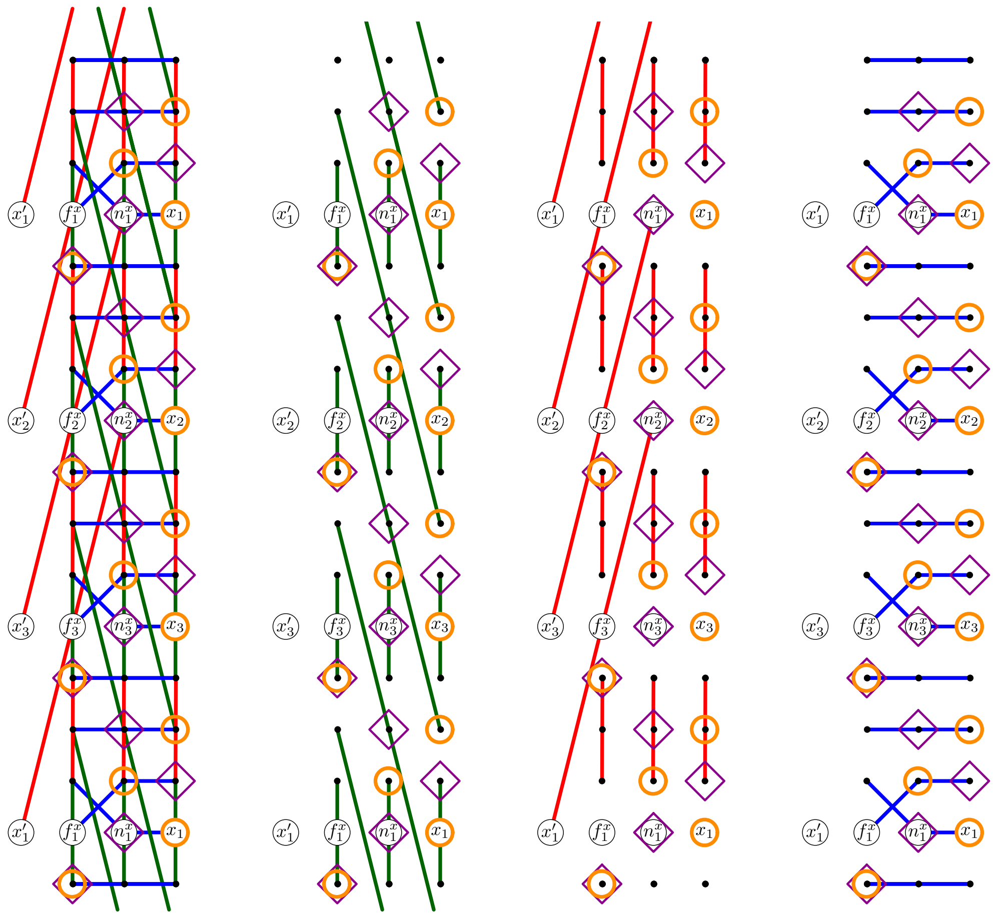


Figure 5d.10: The equality gadget $\psi(x)$, drawn "unrolled". Left to right: hyperedges of all colors; green hyperedges; red hyperedges; blue hyperedges. The two possible IVCs for the gadget are shown as highlighted with orange circles and purple diamonds, respectively.

Figure 5d.11 shows the order in which we may infer other elements of $I$ based on whether $x_1 \in I$ (each vertex is marked in superscript with the step in which an inference about it is made). We implicitly make repeated use of Lemma 5d.3, as briefly described in Figure 5d.7.

**Steps 0-2:** obvious from premise.

**Steps 3-6:** necessarily $I$ contains **either** purple dashed diamond vertices **or** brown dashed circle vertices.

**Steps 7-10:** similarly, $I$ contains **either** pink dashed diamond vertices **or** and cyan dashed circle vertices.

**Steps 11-12:** do not depend on dashed choices.

**Steps 13-14:** depend on dashed assignment for $n_i^x$ and $x_i$, and the fact that $f_i^x$ is always False (i.e., never in $I$).

Suppose $x_1 \in I$. Then all solid orange circle vertices are included in $I$, and because of the diagonal green constraints, we have that inclusion of these vertices entails exclusion of the pink vertices from $I$ (else number 11 and number 7 share a hyperedge). Hence the inclusion of the solid orange circle vertices in $I$ entails the inclusion of the

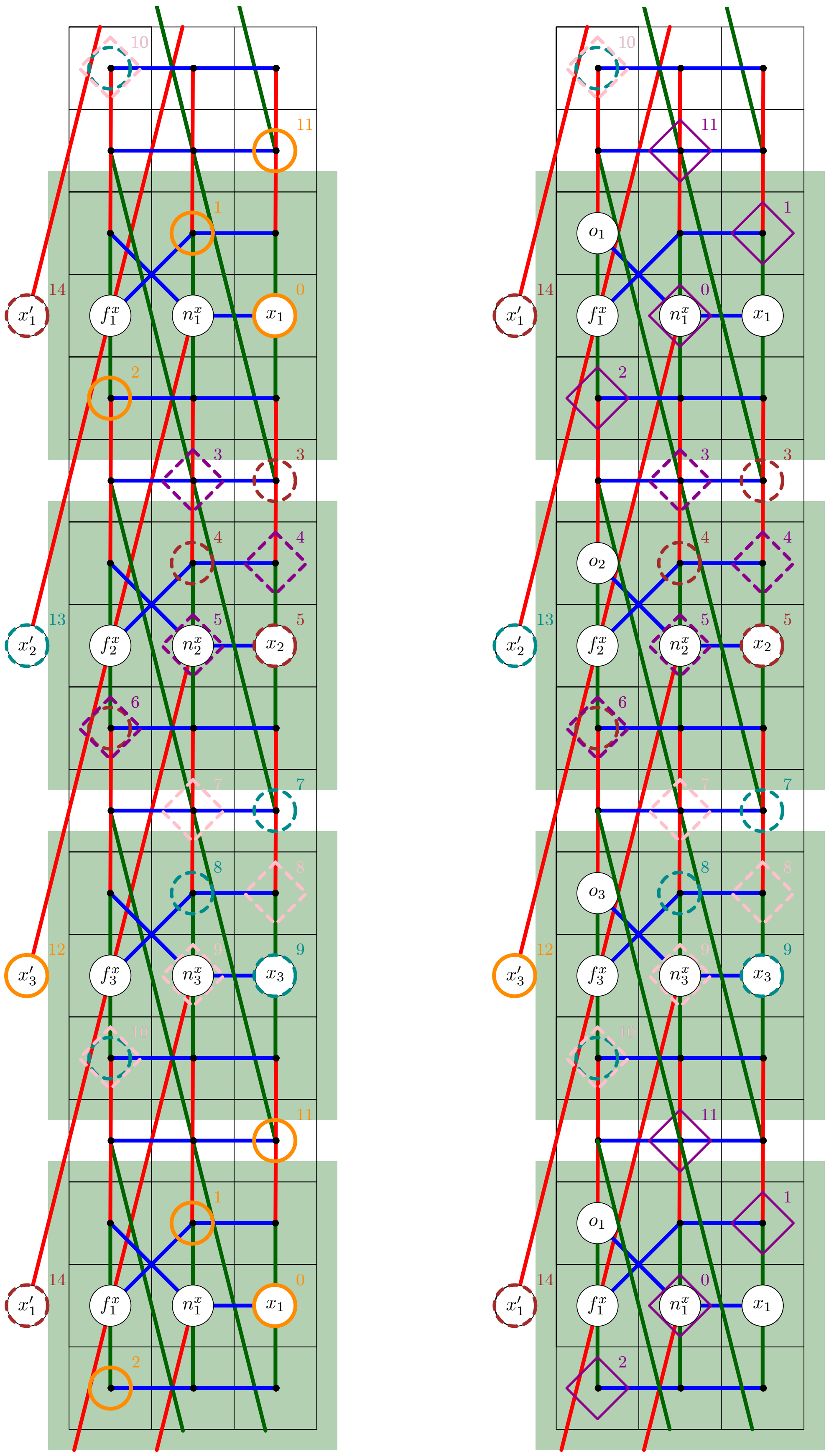


Figure 5d.11: Inferences about elements of $I$ based on whether $x_1 \in I$. Left: $x_1 \in I$, right: $x_1 \notin I$.

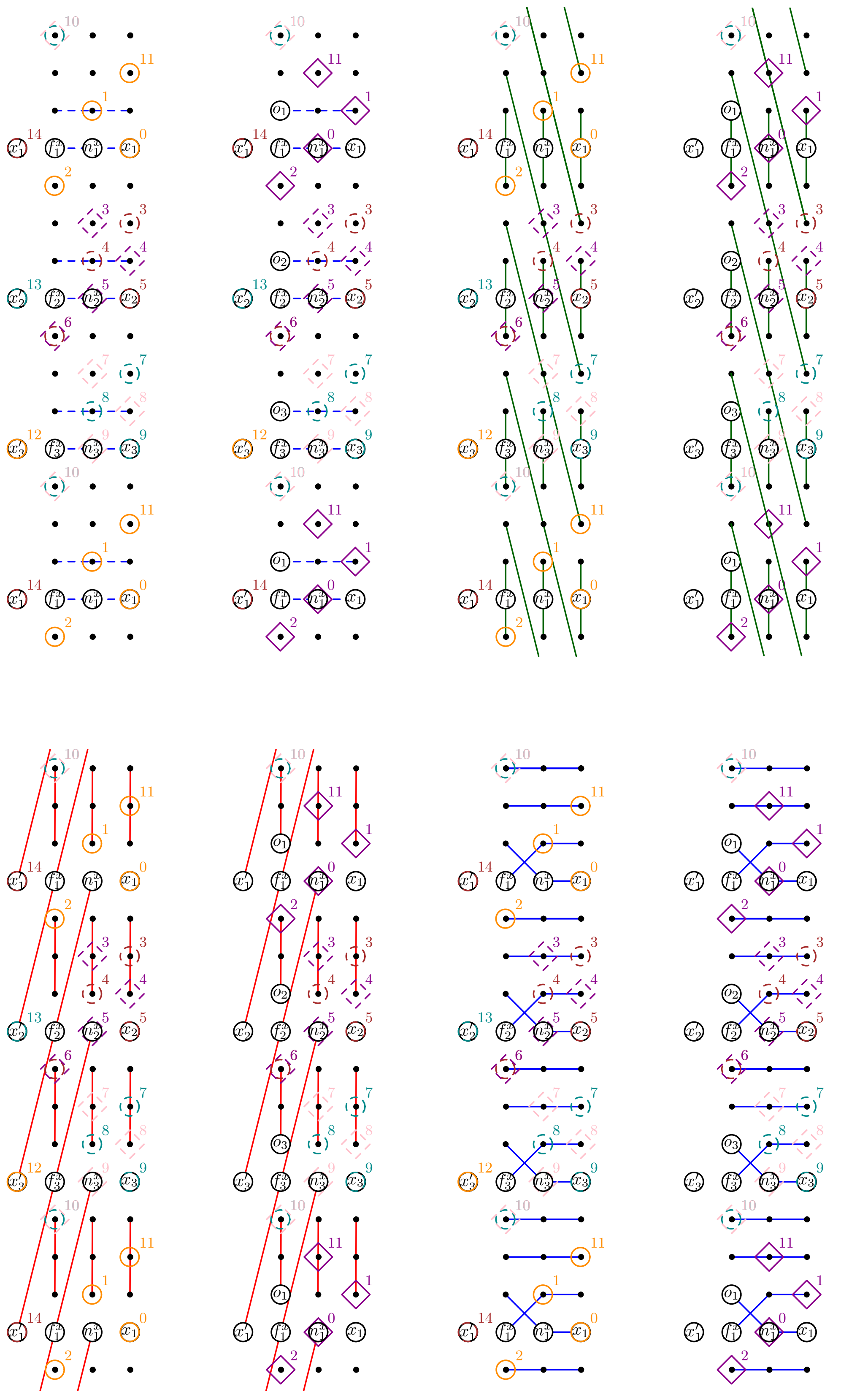


Figure 5d.12: Showing individual hyperedges of Figure 5d.11. Clockwise from top left: dashed "simulated" hyperedges (due to Lemma 5d.3); green hyperedges; red hyperedges; blue hyperedges.

cyan dashed circle vertices. Symmetrically, the inclusion of cyan dashed circle vertices in $I$ entails the inclusion of brown dashed circle vertices. Altogether, then, we have $x_1 \in I \implies$ the entire circle assignment (which is exactly the orange assignment as marked on the equality gadget).

Conversely, suppose instead that $x_1 \notin I$.

Then all solid purple diamond vertices are included in $I$, and because of the diagonal green constraints, we have that inclusion of these vertices entails exclusion of the brown vertices from $I$ (else number 11 and number 3 share a hyperedge). Hence the inclusion of the solid orange circle vertices in $I$ entails the inclusion of the pink dashed diamond vertices. Symmetrically, the inclusion of pink dashed diamond vertices in $I$ entails the inclusion of purple dashed diamond vertices. Altogether, then, we have $x_1 \notin I \implies$ the entire diamond assignment (which is exactly the purple diamond assignment as marked on the equality gadget).

Note we have not (so far) paid particular attention to the vertices $x'_i$. We began with $x_1 \oplus \neg x_1$, and obtained $x_1 = x_2 = x_3$. Additionally, for all $i \in \{1, 2, 3\}$ we have that $x_i = \neg n_i^x$ and $f_i^x = \text{False}$. Note that $x'_i = x_i$ in both the orange assignment and the purple assignment. ■

It remains to deal with our variables $x'_1, x'_2, x'_3$, which appear only in red constraints for now (all other variables introduced so far appear exactly once per color class).

**Balancing colors** For each constraint (or hyperedge) $c = (x, y, z) \in M_\phi$ (recall $M_\phi$ is any perfect matching on $H_\phi$), introduce the three constraints shown in Figure 5d.13, colored blue and green. Applying the fact that $x'_i = x_i$, any assignment satisfying $\phi$ also satisfies the gadget constraints.

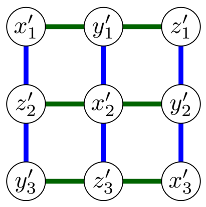


Figure 5d.13: The color-balancing gadget $\eta(c)$

**Tying things together** It should be clear that the constructed formula $\phi'$ obtained by introducing an equality gadget for each variable and a color-balancing gadget for each clause in $M_\phi$ is satisfiable (admits an IVC) if and only if $\phi$ is satisfiable. We now point out that, in addition:

- $\phi'$ is **cubic and tricolor**: every vertex appears in one blue hyperedge, one green hyperedge, and one red hyperedge, **exactly**.
- $\phi'$ is **simple**: the intersection of two constraints is always at most one variable.

We now need only argue that the constructed instance $\phi'$ is **triangle-free**. The miniature sketch of our construction in Figure 5d.14 may be helpful to follow along.

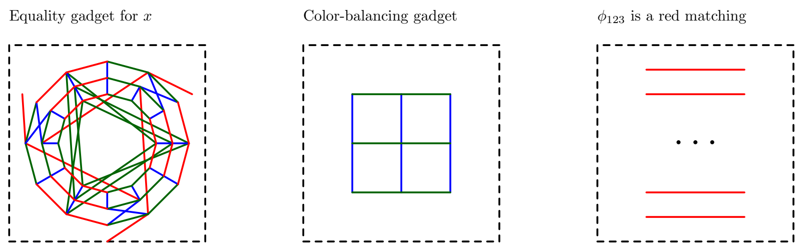


Figure 5d.14: A minimal sketch of the different components of our reduction.

We first point out that $\phi_{123}$, the equality gadget, and the color-balancing gadget are each separately triangle-free (this is immediate for for $\phi_{123}$ and the color-balancing gadget, since they do not contain constraints in all 3 colors). It remains to show that no triangle is introduced when combining them. Suppose for contradiction there is a triangle in the construction. Then necessarily that triangle includes some red hyperedge $(u, v, w)$. However, $(u, v, w)$ cannot be a red hyperedge from $\phi_{123}$, because $u, v, w$ are each copies of different vertices, and appear in equality gadgets for those vertices (variables of $\phi$) together with otherwise nonintersecting blue and green constraints. On the other hand, if $(u, v, w)$ is a red hyperedge from some equality gadget, then only one of $u, v, w$ appears in any hyperedge outside of the equality gadget – so there can be no triangle. We have proven all the desired properties for our instance, and the result follows. □

## 5d.4 Consequence for 3 Dimensional Matching

It quickly follows from this result that a restriction of 3 Dimensional Matching is also NP-complete:

**Corollary 5d.5.** 3 Dimensional Matching is NP-complete even restricted to instances $(X, Y, Z, T)$ such that no three triples in $T$ have pairwise nonempty intersections, and any two triples in $T$ have an intersection of size at most one.

*Proof.* Let $H = (V, \mathcal{E})$ with $\mathcal{E} = R \cup G \cup B$ be (the hypergraph of) a Triangle-free Tricolor Cubic Simple 1-in-3 instance. Note that necessarily, each vertex $v \in V$ is contained in exactly three hyperedges – one of each color. Denote the triple of hyperedges containing a vertex $v$ by $t_v$. Consider the 3 Dimensional Matching instance $(X, Y, Z, T)$ with:

- $X = R$, $Y = G$, and $Z = B$, and
- $T = \{t_v | v \in V\}$

This 3DM instance is a yes-instance if and only if $H$ is a yes-instance of 1-in-3 (i.e., $H$ admits an IVC). Given an IVC $I$ of $H$, the set $M = \{t_v | v \in I\}$ is a perfect 3 dimensional matching, and conversely, given a (necessarily perfect) 3 dimensional matching $M \subsetneq T$, the set $I = \{v | t_v \in M\}$ is an IVC in $H$. Moreover, $H'$ has no triangles (else $H$ would as well) and is simple (else $H$ would not be, either), and the result follows. □

We expect (and hope) that these results will be of use in future works in providing problems to reduce from.

# Epilogue

*D. : Tell me if anything was ever done [3, Note 1365].*

*C. : It's over now [1, Stories of the Street].*

*E. : Thus, he concluded [2, Letter LII, p. 26].*

# Bibliography

[1] Leonard Cohen. Various songs of Leonard Cohen, 1934-2016.

[2] L. Euler. *Lettres à une princesse d'Allemagne, sur divers sujets de physique et de philosophie*, volume 1. Hachette, 1842. URL https://archive.org/details/lettresdeleuleru01eule.

[3] L. Da Vinci. *The Notebooks of Leonardo Da Vinci.* The Notebooks of Leonardo Da Vinci. Project Gutenberg, 1888. URL https://www.gutenberg.org/ebooks/5000. Translation by Jean-Paul Richter.

[4] L. Euler. *Lettres à une princesse d'Allemagne, sur divers sujets de physique et de philosophie*, volume 2. Hachette, 1842. URL https://archive.org/details/lettresdeleuleru02eule.

[5] OpenStreetMap contributors. Screenshot of Manhattan and surroundings retrieved from https://www.openstreetmap.org . https://www.openstreetmap.org, 2025.

[6] Teo F. Kutner. Personal Communication, 2025.

[7] David Smith. So you thought Sudoku came from the Land of the Rising Sun ... *The Guardian*, 2005. URL https://www.theguardian.com/media/2005/may/15/pressandpublishing.usnews.

[8] a-kat.com. Sudokusolver, 2008. URL http://www.a-kat.com/programming/cpp/sudoku/sudoku.html. Accessible through the internet archive.

[9] Takayuki Yato and Takahiro Seta. Complexity and completeness of finding another solution and its application to puzzles. *IEICE transactions on fundamentals of electronics, communications and computer sciences*, 86(5):1052–1060, 2003.

[10] Richard M. Karp. Reducibility among combinatorial problems. In Raymond E. Miller and James W. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, New York, 1972.

[11] Larry Eisenberg and Thomas H Noe. Systemic risk in financial systems. *Management Science*, 47(2):236–249, 2001.

[12] Beni Egressy and Roger Wattenhofer. Bailouts in financial networks. *CoRR*, abs/2106.12315, 2021. URL https://arxiv.org/abs/2106.12315.

[13] Nils Bertschinger, Martin Hoefer, and Daniel Schmand. Flow allocation games. *Mathematics of Operations Research*, 2024.

[14] Pál András Papp and Roger Wattenhofer. Sequential defaulting in financial networks. In *12th Innovations in Theoretical Computer Science Conference, ITCS*, volume 185 of *LIPIcs*, 2021.

[15] BBC, 2010. Doctor Who: Series 5.

[16] Eleni C. Akrida, George B. Mertzios, and Paul G. Spirakis. The Temporal Explorer Who Returns to the Base. In Pinar Heggernes, editor, *Algorithms and Complexity*, Lecture Notes in Computer Science, pages 13–24, Cham, 2019. Springer International Publishing. ISBN 978-3-030-17402-6. doi: 10.1007/978-3-030-17402-6_2.

[17] Davide Bilò, Gianlorenzo D'Angelo, Luciano Gualà, Stefano Leucci, and Mirko Rossi. Blackout-Tolerant Temporal Spanners. In Thomas Erlebach and Michael Segal, editors, *Algorithmics of Wireless Networks*, Lecture Notes in Computer Science, pages 31–44, Cham, 2022. Springer International Publishing. ISBN 978-3-031-22050-0. doi: 10.1007/978-3-031-22050-0_3.

[18] Arnaud Casteigts, Joseph G. Peters, and Jason Schoeters. Temporal cliques admit sparse spanners. *Journal of Computer and System Sciences*, 121:1–17, November 2021. ISSN 0022-0000. doi: 10.1016/j.jcss.2021.04.004.

[19] Eugen Füchsle, Hendrik Molter, Rolf Niedermeier, and Malte Renken. Delay-Robust Routes in Temporal Graphs, January 2022. arXiv:2201.05390 [cs].

[20] Andrea Marino and Ana Silva. Eulerian walks in temporal graphs. *Algorithmica*, 85(3):805–830, 2023. doi: 10.1007/S00453-022-01021-Y. URL https://doi.org/10.1007/s00453-022-01021-y.

[21] Alfredo Braunstein and Alessandro Ingrosso. Inference of causality in epidemics on temporal contact networks. *Scientific Reports*, 6(1):27538, June 2016. ISSN 2045-2322. doi: 10.1038/srep27538.

[22] Jessica Enright, Kitty Meeks, George B. Mertzios, and Viktor Zamaraev. Deleting edges to restrict the size of an epidemic in temporal networks. *Journal of Computer and System Sciences*, 119:60–77, August 2021. ISSN 0022-0000. doi: 10.1016/j.jcss.2021.01.007.

[23] Jessica Enright, Kitty Meeks, and Fiona Skerman. Assigning times to minimise reachability in temporal graphs. *Journal of Computer and System Sciences*, 115:169–186, February 2021. ISSN 0022-0000. doi: 10.1016/j.jcss.2020.08.001.

[24] Eugenio Valdano, Luca Ferreri, Chiara Poletto, and Vittoria Colizza. Analytical Computation of the Epidemic Threshold on Temporal Networks. *Physical Review X*, 5(2):021005, April 2015. doi: 10.1103/PhysRevX.5.021005.

[25] John Tang, Ilias Leontiadis, Salvatore Scellato, Vincenzo Nicosia, Cecilia Mascolo, Mirco Musolesi, and Vito Latora. Applications of Temporal Graph Metrics to Real-World Networks. In Petter Holme and Jari Saramäki, editors, *Temporal Networks*, Understanding Complex Systems, pages 135–159. Springer, Berlin, Heidelberg, 2013. ISBN 978-3-642-36461-7. doi: 10.1007/978-3-642-36461-7_7.

[26] Richard T. Bumby. A Problem with Telephones. *SIAM Journal on Algebraic Discrete Methods*, 2(1):13–18, March 1981. ISSN 0196-5212. doi: 10.1137/0602002. URL https://epubs.siam.org/doi/abs/10.1137/0602002. Publisher: Society for Industrial and Applied Mathematics.

[27] George B. Mertzios, Hendrik Molter, and Paul G. Spirakis. Realizing temporal transportation trees, March 2024. URL http://arxiv.org/abs/2403.18513. arXiv:2403.18513 [cs].

[28] Davi de Andrade, Júlio Araújo, Allen Ibiapina, Andrea Marino, Jason Schoeters, and Ana Silva. Temporal cycle detection and acyclic temporization, 2025. URL https://arxiv.org/abs/2503.02694.

[29] Thomas Erlebach, Nils Morawietz, and Petra Wolf. Parameterized Algorithms for Multi-Label Periodic Temporal Graph Realization. In Arnaud Casteigts and Fabian Kuhn, editors, *3rd Symposium on Algorithmic Foundations of Dynamic Networks (SAND 2024)*, volume 292 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 12:1–12:16, Dagstuhl, Germany, 2024. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. ISBN 978-3-95977-315-7. doi: 10.4230/LIPIcs.SAND.2024.12. ISSN: 1868-8969.

[30] Mark Cartwright and Ubisoft Entertainment SA. Lighthouse of alexandria. *Ancient History Encyclopedia*, 2018.

[31] Kendall Murphy and Katherine Schauer. NASA's First Two-way End-to-End Laser Communications Relay System, 2023. URL https://www.nasa.gov/technology/space-comms/nasas-first-two-way-end-to-end-laser-communications-system/.

[32] Patrick L. Thompson, Armen Caroglanian, Jeffrey A. Guzek, Stephen A Hall, Robert E. Lafon, Kristoffer C. Olsen, Daniel A. Paulson, Haleh Safavi, Predrag Sekulic, Oscar Ta, and Mark E. Wilson. NASA's LCOT (low-cost optical terminal) FSOS (free-space optical subsystem): concept, design, build, and test. In Hamid Hemmati and Bryan S. Robinson, editors, *Free-Space Laser Communications XXXV*, volume 12413, page 124130X. International Society for Optics and Photonics, SPIE, 2023. doi: 10.1117/12.2653355. URL https://doi.org/10.1117/12.2653355.

[33] Charidimos Chaintoutis, Behnam Shariati, Adonis Bogris, Paul V. Dijk, Chris G. H. Roeloffzen, Jerome Bourderionnet, Ioannis Tomkos, and Dimitris Syvridis.

Free space intra-datacenter interconnects based on 2d optical beam steering enabled by photonic integrated circuits. *Photonics*, 5(3), 2018. ISSN 2304-6732. doi: 10.3390/photonics5030021. URL https://www.mdpi.com/2304-6732/5/3/21.

[34] Navid Hamedazimi, Zafar Qazi, Himanshu Gupta, Vyas Sekar, Samir R. Das, Jon P. Longtin, Himanshu Shah, and Ashish Tanwer. Firefly: a reconfigurable wireless data center fabric using free-space optics. In *Proceedings of the 2014 ACM Conference on SIGCOMM*, SIGCOMM '14, page 319–330, New York, NY, USA, 2014. Association for Computing Machinery. ISBN 9781450328364. doi: 10.1145/2619239.2626328. URL https://doi.org/10.1145/2619239.2626328.

[35] Shaojuan Zhang, Xuwei Xue, Eduward Tangdiongga, and Nicola Calabretta. Low-latency optical wireless data-center networks using nanoseconds semiconductor-based wavelength selectors and arrayed waveguide grating router. *Photonics*, 9(3), 2022. ISSN 2304-6732. doi: 10.3390/photonics9030203. URL https://www.mdpi.com/2304-6732/9/3/203.

[36] Thomas Fenz, Klaus-Tycho Foerster, Stefan Schmid, and Anaïs Villedieu. Efficient non-segregated routing for reconfigurable demand-aware networks. *Comput. Commun.*, 164:138–147, 2020.

[37] Klaus-Tycho Foerster, Manya Ghobadi, and Stefan Schmid. Characterizing the algorithmic complexity of reconfigurable data center architectures. In *Proc. of Symp. on Architectures for Networking and Communications Systems (ANCS)*, pages 89–96. ACM Press, 2018.

[38] Klaus-Tycho Foerster, Maciej Pacut, and Stefan Schmid. On the complexity of non-segregated routing in reconfigurable data center architectures. *Comput. Commun. Rev.*, 49:2–81, 2019.

[39] Mohammad Al-Fares, Alexander Loukissas, and Amin Vahdat. A scalable, commodity data center network architecture. *ACM SIGCOMM Comput. Commun. Rev.*, 38:63–74, 2008.

[40] Chuanxiong Guo, Guohan Lu, Dan Li, Haitao Wu, Xuan Zhang, Yunfeng Shi, Chen Tian, Yongguang Zhang, and Songwu Lu. BCube: A high performance, server-centric network architecture for modular data centers. *ACM SIGCOMM Comput. Commun. Rev.*, 39:63–74, 2009.

[41] Yehuda Afek, Noga Alon, Omer Barad, Eran Hornstein, Naama Barkai, and Ziv Bar-Joseph. A biological solution to a fundamental distributed computing problem. *Science*, 331(6014):183–185, 2011. doi: 10.1126/science.1193210. URL https://www.science.org/doi/abs/10.1126/science.1193210.

[42] Joffroy Beauquier, Janna Burman, Fabien Dufoulon, and Shay Kutten. Fast Beeping Protocols for Deterministic MIS and $(\Delta + 1)$-Coloring in Sparse

Graphs. In *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications*, pages 1754–1762, 2018. doi: 10.1109/INFOCOM.2018.8486015.

[43] Mohsen Ghaffari. Local computation of maximal independent set. In *2022 IEEE 63rd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 438–449, 2022. doi: 10.1109/FOCS54457.2022.00049.

[44] Tom Friedetzky, David C. Kutner, George B. Mertzios, Iain A. Stewart, and Amitabh Trehan. Payment scheduling in the interval debt model. *Theoretical Computer Science*, 1028:115028, 2025. ISSN 0304-3975. doi: 10.1016/j.tcs.2024.115028.

[45] Tom Friedetzky, David C. Kutner, George B. Mertzios, Iain A. Stewart, and Amitabh Trehan. Payment scheduling in the interval debt model. *SOFSEM 2023: Theory and Practice of Computer Science LNCS 13878*, page 267, 2023.

[46] David C. Kutner and Laura Larios-Jones. Temporal reachability dominating sets: contagion in temporal graphs. *Journal of Computer and System Sciences*, page 103701, 2025. ISSN 0022-0000. URL https://doi.org/10.1016/j.jcss.2025.103701.

[47] David C. Kutner and Laura Larios-Jones. Temporal reachability dominating sets: Contagion in temporal graphs. In Konstantinos Georgiou and Evangelos Kranakis, editors, *Algorithmics of Wireless Networks*, pages 101–116, Cham, 2023. Springer Nature Switzerland. ISBN 978-3-031-48882-5.

[48] David C. Kutner and Iain A. Stewart. Reconfigurable routing in data center networks. *Theoretical Computer Science*, page 115154, 2025. ISSN 0304-3975. URL https://doi.org/10.1016/j.tcs.2025.115154.

[49] David C. Kutner and Iain A. Stewart. Reconfigurable routing in data center networks. In *Algorithmics of Wireless Networks: 20th International Symposium, ALGOWIN 2024, Egham, UK, September 5–6, 2024, Proceedings*, page 117–130, Berlin, Heidelberg, 2024. Springer-Verlag. ISBN 978-3-031-74579-9. URL https://doi.org/10.1007/978-3-031-74580-5_9.

[50] Maximilien Gadouleau and David C. Kutner. Generalising the maximum independent set algorithm via Boolean networks, March 2024. URL http://arxiv.org/abs/2403.17658. arXiv:2403.17658 [cs].

[51] David C. Kutner and Anouk Sommer. Better Late, Then? The Hardness of Choosing Delays to Meet Passenger Demands in Temporal Graphs. In Kitty Meeks and Christian Scheideler, editors, *4th Symposium on Algorithmic Foundations of Dynamic Networks (SAND 2025)*, volume 330 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 7:1–7:18, Dagstuhl, Germany, 2025. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. ISBN 978-3-95977-368-3. URL https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.SAND.2025.7.

[52] Leonard C.G. Rogers and Luitgard A.M. Veraart. Failure and rescue in an interbank network. *Management Science*, 59(4):882–898, 2013.

[53] Steffen Schuldenzucker, Sven Seuken, and Stefano Battiston. Finding clearing payments in financial networks with credit default swaps is PPAD-complete. In *8th Innovations in Theoretical Computer Science Conference, ITCS*, volume 67 of *LIPIcs*, 2017.

[54] Pál András Papp and Roger Wattenhofer. Network-Aware Strategies in Financial Systems. In *ICALP 2020*, volume 168, 2020. ISBN 978-3-95977-138-2. doi: 10.4230/LIPIcs.ICALP.2020.91.

[55] Panagiotis Kanellopoulos, Maria Kyropoulou, and Hao Zhou. Forgiving debt in financial network games. In *IJCAI-22*, 7 2022. doi: 10.24963/ijcai.2022/48.

[56] Panagiotis Kanellopoulos, Maria Kyropoulou, and Hao Zhou. On priority-proportional payments in financial networks. *Theoretical Computer Science*, 1014:114767, 2024. ISSN 0304-3975. doi: https://doi.org/10.1016/j.tcs.2024.114767. URL https://www.sciencedirect.com/science/article/pii/S0304397524003840.

[57] Andrew G Haldane and Robert M May. Systemic risk in banking ecosystems. *Nature*, 469(7330):351–355, 2011.

[58] Marco Bardoscia, Paolo Barucca, Stefano Battiston, Fabio Caccioli, Giulio Cimini, Diego Garlaschelli, Fabio Saracco, Tiziano Squartini, and Guido Caldarelli. The physics of financial networks. *Nature Reviews Physics*, 3(7):490–507, 2021.

[59] Larry Eisenberg. A summary: Boolean networks applied to systemic risk. *Neural Networks in Financial Engineering*, pages 436–449, 1996.

[60] Jean-Charles Rochet and Xavier Vives. Coordination failures and the lender of last resort: was Bagehot right after all? *Journal of the European Economic Association*, 2(6):1116–1147, 2004.

[61] Walter Bagehot. *Lombard street: a description of the money market*. HS King & Company, London, 1873.

[62] Marios Papachristou and Jon Kleinberg. Allocating stimulus checks in times of crisis. In *Proceedings of the ACM Web Conference 2022*, pages 16–26, 2022.

[63] Bezaye Tesfaye, Nikolaus Augsten, Mateusz Pawlik, Michael Böhlen, and Christian Jensen. Speeding up reachability queries in public transport networks using graph partitioning. *Information Systems Frontiers*, 24:11–29, 2022.

[64] P. Holme and J. Saramäki, editors. *Temporal Networks*. Springer, London, 2013.

[65] Othon Michail. An introduction to temporal graphs: An algorithmic perspective. *Internet Mathematics*, 12(4):239–280, 2016.

[66] David Kempe, Jon M. Kleinberg, and Amit Kumar. Connectivity and inference problems for temporal networks. In *Proceedings of the 32nd annual ACM symposium on Theory of computing (STOC)*, pages 504–513, 2000.

[67] Nina Klobas, George B. Mertzios, Hendrik Molter, Rolf Niedermeier, and Philipp Zschoche. Interference-free walks in time: Temporally disjoint paths. In *Proceedings of the 30th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 4090–4096, 2021.

[68] Eleni C. Akrida, George B. Mertzios, Paul G. Spirakis, and Viktor Zamaraev. Temporal vertex cover with a sliding time window. *Journal of Computer and System Sciences*, 107:108–123, February 2020. ISSN 0022-0000. doi: 10.1016/j.jcss.2019.08.002.

[69] Audience. Audience participation at the Cambridge LIPNE Workshop on Computational complexity and economic decision making, 2024.

[70] Craig A. Tovey. A simplified NP-complete satisfiability problem. *Discrete Applied Mathematics*, 8:85–89, 1984.

[71] M.R. Garey and D.S. Johnson, editors. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman and Company, United States, 1979.

[72] Argyrios Deligkas and Igor Potapov. Optimizing reachability sets in temporal graphs by delaying. *Information and Computation*, 285:104890, May 2022. ISSN 0890-5401. doi: 10.1016/j.ic.2022.104890.

[73] Hendrik Molter, Malte Renken, and Philipp Zschoche. Temporal reachability minimization: Delaying vs. deleting. *arXiv preprint arXiv:2102.10814*, 2021.

[74] Nina Klobas, George B. Mertzios, Hendrik Molter, and Paul G. Spirakis. The Complexity of Computing Optimum Labelings for Temporal Connectivity. In Stefan Szeider, Robert Ganian, and Alexandra Silva, editors, *47th International Symposium on Mathematical Foundations of Computer Science (MFCS 2022)*, volume 241 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 62:1–62:15, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. ISBN 978-3-95977-256-3. doi: 10.4230/LIPIcs.MFCS.2022.62. URL https://drops.dagstuhl.de/opus/volltexte/2022/16860.

[75] Thomas Erlebach and Jakob T. Spooner. A game of cops and robbers on graphs with periodic edge-connectivity. *CoRR*, 2019.

[76] Thomas Erlebach and Jakob T. Spooner. Faster Exploration of Degree-Bounded Temporal Graphs. In Igor Potapov, Paul Spirakis, and James Worrell, editors, *43rd International Symposium on Mathematical Foundations of Com-*

*puter Science (MFCS 2018)*, volume 117 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 36:1–36:13, Dagstuhl, Germany, 2018. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. ISBN 978-3-95977-086-6. doi: 10.4230/LIPIcs.MFCS.2018.36.

[77] John Whitbeck, Marcelo Dias de Amorim, Vania Conan, and Jean-Loup Guillaume. Temporal reachability graphs. In *Proceedings of the 18th Annual International Conference on Mobile Computing and Networking*, Mobicom '12, page 377–388, New York, NY, USA, 2012. Association for Computing Machinery. ISBN 9781450311595. URL https://doi.org/10.1145/2348543.2348589.

[78] Arnaud Casteigts, Timothée Corsini, and Writika Sarkar. Simple, strict, proper, happy: A study of reachability in temporal graphs. *Theoretical Computer Science*, 991:114434, April 2024. ISSN 0304-3975. doi: 10.1016/j.tcs.2024.114434. URL https://www.sciencedirect.com/science/article/pii/S0304397524000495.

[79] Stefan Balev, Eric Sanlaville, and Jason Schoeters. Temporally connected components. *Theoretical Computer Science*, 1013:114757, 2024. ISSN 0304-3975. doi: https://doi.org/10.1016/j.tcs.2024.114757. URL https://www.sciencedirect.com/science/article/pii/S0304397524003748.

[80] Sandeep Bhadra and Afonso Ferreira. Complexity of Connected Components in Evolving Graphs and the Computation of Multicast Trees in Dynamic Networks. In Samuel Pierre, Michel Barbeau, and Evangelos Kranakis, editors, *Ad-Hoc, Mobile, and Wireless Networks*, Lecture Notes in Computer Science, pages 259–270, Berlin, Heidelberg, 2003. Springer. ISBN 978-3-540-39611-6. doi: 10.1007/978-3-540-39611-6_23.

[81] Esteban Christiann, Eric Sanlaville, and Jason Schoeters. On inefficiently connecting temporal networks. *arXiv preprint arXiv:2312.07117*, 2023.

[82] Arnaud Casteigts. *A Journey through Dynamic Networks (with Excursions)*. Thesis, Université de Bordeaux, June 2018. URL https://hal.science/tel-01883384.

[83] O. Pybus, A. Rambaut, COG-UK-Consortium, et al. Preliminary analysis of SARS-CoV-2 importation & establishment of UK transmission lineages. *Virological. org.*, 2020.

[84] Argyrios Deligkas, Eduard Eiben, and George Skretas. Minimizing reachability times on temporal graphs via shifting labels. In Edith Elkind, editor, *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence, IJCAI-23*, pages 5333–5340, Macao, 8 2023. doi: 10.24963/ijcai.2023/592.

[85] Jessica Enright, Laura Larios-Jones, Kitty Meeks, and William Pettersson. Reachability in temporal graphs under perturbation. *SOFSEM 2025: Theory and Practice of Computer Science LNCS*, 2025.

[86] Randall Munroe. xkcd: Post Vaccine Social Scheduling, 2021. URL https://xkcd.com/2450/.

[87] Hiroshi Eto, Fengrui Guo, and Eiji Miyano. Distance-$d$ independent set problems for bipartite and chordal graphs. *Journal of Combinatorial Optimization*, 27(1):88–99, January 2014. ISSN 1573-2886. doi: 10.1007/s10878-012-9594-4.

[88] Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer International Publishing, Cham, 2015. ISBN 978-3-319-21274-6 978-3-319-21275-3. doi: 10.1007/978-3-319-21275-3.

[89] Bruno Courcelle. The expression of graph properties and graph transformations in monadic second-order logic. In *Handbook Of Graph Grammars And Computing By Graph Transformation: Volume 1: Foundations*, pages 313–400. World Scientific, Singapore, 1997.

[90] David Peleg. Time-efficient broadcasting in radio networks: A review. In *International Conference on Distributed Computing and Internet Technology*, pages 1–18. Springer, 2007.

[91] Thomas Erlebach and Alexander Hall. NP-Hardness of Broadcast Scheduling and Inapproximability of Single-Source Unsplittable Min-Cost Flow. *Journal of Scheduling*, 7(3):223–241, May 2004. ISSN 1099-1425. doi: 10.1023/B:JOSH.0000019682.75022.96.

[92] Andreas Jakoby, Rüdiger Reischuk, and Christian Schindelhauer. The complexity of broadcasting in planar and decomposable graphs. In Ernst W. Mayr, Gunther Schmidt, and Gottfried Tinhofer, editors, *Graph-Theoretic Concepts in Computer Science*, Lecture Notes in Computer Science, pages 219–231, Berlin, Heidelberg, 1995. Springer. ISBN 978-3-540-49183-5. doi: 10.1007/3-540-59071-4_50.

[93] Arnaud Casteigts, Paola Flocchini, Walter Quattrociocchi, and Nicola Santoro. Time-Varying Graphs and Dynamic Networks. In Hannes Frey, Xu Li, and Stefan Ruehrup, editors, *Ad-hoc, Mobile, and Wireless Networks*, Lecture Notes in Computer Science, pages 346–359, Berlin, Heidelberg, 2011. Springer. ISBN 978-3-642-22450-8. doi: 10.1007/978-3-642-22450-8_27.

[94] Ma-Lian Chia, David Kuo, and Mei-Feng Tung. The multiple originator broadcasting problem in graphs. *Discrete Applied Mathematics*, 155(10):1188–1199, May 2007. ISSN 0166-218X. doi: 10.1016/j.dam.2006.10.011.

[95] Hayk Grigoryan. *Problems related to broadcasting in graphs.* phd, Concordia University, September 2013. URL https://spectrum.library.concordia.ca/id/eprint/977773/.

[96] Benjamin Merlin Bumpus and Kitty Meeks. Edge exploration of temporal graphs. In Paola Flocchini and Lucia Moura, editors, *Combinatorial Algorithms*, Lecture Notes in Computer Science, pages 107–121, Cham, 2021. Springer. ISBN 978-3-030-79987-8. doi: 10.1007/978-3-030-79987-8_8.

[97] George B. Mertzios, Hendrik Molter, Rolf Niedermeier, Viktor Zamaraev, and Philipp Zschoche. Computing maximum matchings in temporal graphs. *Journal of Computer and System Sciences*, 137:1–19, November 2023. ISSN 0022-0000. doi: 10.1016/j.jcss.2023.04.005.

[98] Nina Klobas, George B. Mertzios, Hendrik Molter, Rolf Niedermeier, and Philipp Zschoche. Interference-free walks in time: Temporally disjoint paths. *Autonomous Agents and Multi-Agent Systems*, 37(1):1, 2023.

[99] Jessica Enright and Kitty Meeks. Deleting Edges to Restrict the Size of an Epidemic: A New Application for Treewidth. *Algorithmica*, 80(6):1857–1889, June 2018. ISSN 1432-0541. doi: 10.1007/s00453-017-0311-7.

[100] Thomas Erlebach, Michael Hoffmann, and Frank Kammer. On Temporal Graph Exploration. In Magnús M. Halldórsson, Kazuo Iwama, Naoki Kobayashi, and Bettina Speckmann, editors, *Automata, Languages, and Programming*, Lecture Notes in Computer Science, pages 444–455, Berlin, Heidelberg, 2015. Springer. ISBN 978-3-662-47672-7. doi: 10.1007/978-3-662-47672-7_36.

[101] Roman Haag, Hendrik Molter, Rolf Niedermeier, and Malte Renken. Feedback edge sets in temporal graphs. *Discrete Applied Mathematics*, 307:65–78, January 2022. ISSN 0166-218X. doi: 10.1016/j.dam.2021.09.029.

[102] Bruno Courcelle and Joost Engelfriet. Monadic second-order logic. In Bruno Courcelle and Joost Engelfriet, editors, *Graph Structure and Monadic Second-Order Logic: A Language-Theoretic Approach*, Encyclopedia of Mathematics and its Applications, pages 315–426. Cambridge University Press, Cambridge, 2012. ISBN 978-0-521-89833-1. doi: 10.1017/CBO9780511977619.007.

[103] Rod G. Downey and Michael R. Fellows. Fixed-Parameter Tractability and Completeness I: Basic Results. *SIAM Journal on Computing*, 24(4):873–921, August 1995. ISSN 0097-5397. doi: 10.1137/S0097539792228228.

[104] Andreas Björklund, Thore Husfeldt, Petteri Kaski, and Mikko Koivisto. Fourier meets Möbius: fast subset convolution. *arXiv e-prints*, page cs/0611101, November 2006. doi: 10.48550/arXiv.cs/0611101.

[105] Hiroshi Eto, Takehiro Ito, Zhilong Liu, and Eiji Miyano. Approximability of the Distance Independent Set Problem on Regular Graphs and Planar Graphs. In T-H. Hubert Chan, Minming Li, and Lusheng Wang, editors, *Combinatorial Optimization and Applications*, Lecture Notes in Computer Science, pages 270–284, Cham, 2016. Springer International Publishing. ISBN 978-3-319-48749-6. doi: 10.1007/978-3-319-48749-6_20.

[106] Geir Agnarsson, Peter Damaschke, and Magnús M. Halldórsson. Powers of geometric intersection graphs and dispersion algorithms. *Discrete Applied Mathematics*, 132(1):3–16, October 2003. ISSN 0166-218X. doi: 10.1016/S0166-218X(03)00386-X.

[107] M. Verheije. *Algorithms for Domination Problems on Temporal Graphs*. PhD thesis, Utrecht University, 2021. URL https://studenttheses.uu.nl/handle/20.500.12932/41240. Accepted: 2021-08-26.

[108] Tom Davot, Jessica Enright, and Laura Larios-Jones. Parameterised algorithms for temporal reconfiguration problems. *arXiv preprint arXiv:2502.11961*, 2025.

[109] Eleni C Akrida, Leszek Gąsieniec, George B Mertzios, and Paul G Spirakis. Ephemeral networks with random availability of links: The case of fast networks. *Journal of Parallel and Distributed Computing*, 87:109–120, 2016.

[110] George B Mertzios, Othon Michail, Ioannis Chatzigiannakis, and Paul G Spirakis. Temporal network optimization subject to connectivity constraints. In *International Colloquium on Automata, Languages, and Programming*, pages 657–668. Springer, 2013.

[111] M. R. Garey, D. S. Johnson, and L. Stockmeyer. Some simplified NP-complete problems. In *Proceedings of the sixth annual ACM symposium on Theory of computing*, STOC '74, pages 47–63, New York, NY, USA, April 1974. Association for Computing Machinery. ISBN 978-1-4503-7423-1. doi: 10.1145/800119.803884.

[112] Simon Tippenhauer and Wolfgang Muzler. On planar 3-sat and its variants. *Fachbereich Mathematik und Informatik der Freien Universitat Berlin*, 2016.

[113] Sanjeev Arora and Boaz Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, Cambridge, 2009. ISBN 978-0-521-42426-4. doi: 10.1017/CBO9780511804090.

[114] Christos H. Papadimitriou. *Computational complexity*. Addison-Wesley, Reading, Mass, 1994. ISBN 978-0-201-53082-7.

[115] Ian Holyer. The NP-completeness of edge-coloring. *SIAM Journal on computing*, 10(4):718–720, 1981.

[116] Shai Gutner. The complexity of planar graph choosability, February 2008. arXiv:0802.2668 [cs].

[117] Rolf Niedermeier. *Invitation to Fixed-Parameter Algorithms*. Oxford University Press, England, February 2006. ISBN 978-0-19-171391-0. doi: 10.1093/acprof:oso/9780198566076.001.0001.

[118] Hans L. Bodlaender. A Linear-Time Algorithm for Finding Tree-Decompositions of Small Treewidth. *SIAM Journal on Computing*, 25(6):1305–1317, December 1996. ISSN 0097-5397. doi: 10.1137/S0097539793251219.

[119] Stefan Arnborg, Jens Lagergren, and Detlef Seese. Easy problems for tree-decomposable graphs. *Journal of Algorithms*, 12(2):308–340, June 1991. ISSN 0196-6774. doi: 10.1016/0196-6774(91)90006-K.

[120] Alexander Langer, Felix Reidl, Peter Rossmanith, and Somnath Sikdar. Practical algorithms for MSO model-checking on tree-decomposable graphs. *Computer Science Review*, 13-14:39–74, November 2014. ISSN 1574-0137. doi: 10.1016/j.cosrev.2014.08.001.

[121] Joachim Kneis, Daniel Mölle, and Peter Rossmanith. Partial vs. Complete Domination: t-Dominating Set. In Jan van Leeuwen, Giuseppe F. Italiano, Wiebe van der Hoek, Christoph Meinel, Harald Sack, and František Plášil, editors, *SOFSEM 2007: Theory and Practice of Computer Science*, Lecture Notes in Computer Science, pages 367–376, Berlin, Heidelberg, 2007. Springer. ISBN 978-3-540-69507-3. doi: 10.1007/978-3-540-69507-3_31.

[122] Tao Chen, Xiaofeng Gao, and Chen Guihai. The features, hardware, and architectures of data center networks: a survey. *J. Parallel Distrib. Comput.*, 96: 45–74, 2016.

[123] Matthew Nance Hall, Kalsu-Tycho Foerster, Stefan Schmid, and Ramakrishnan Durairajan. A survey of reconfigurable optical networks. *Opt. Switch. Netw.*, 41:100621, 2021.

[124] Chan Avin and Stefan Schmid. Toward demand-aware networking: a theory for self-adjusting networks. *ACM SIGCOMM Comput. Commun. Rev.*, 48:31–40, 2019.

[125] Klaus-Tycho Foerster and Stefan Schmid. Survey of reconfigurable data center networks: enablers, algorithms, complexity. *ACM SIGACT News*, 50:62–79, 2019.

[126] Chuanxiong Guo, Haitao Wu, Kun Tan, Lei Shi, Yongguang Zhang, and Songwu Lu. DCell: a scalable and fault-tolerant network structure for data centers. In *Proc. of ACM SIGCOMM Conf. on Data Communication*, pages 75–86, 2008.

[127] Ankit Singla, Chi-Yao Hong, Lucian Popa, and P. Brighten Godfrey. Jellyfish: networking data centers randomly. In *Proc. of 9th USENIX Conf. on Networked Systems Design and Implementation*, pages 225–238, 2012.

[128] Asaf Valadarsky, Gal Shahaf, Michael Dinitz, and Michael Schapira. Xpander: towards optimal-performance datacenters. In *Proc. of 12th Int. Conf. on Emerging Networking Experiments and Technologies*, pages 205–219, 2016.

[129] Igor Pak and Radoš Radoičić. Hamiltonian paths in cayley graphs. *Discrete Mathematics*, 309(17):5501–5508, 2009. ISSN 0012-365X. doi: https://doi.org/10.1016/j.disc.2009.02.018. URL https://www.sciencedirect.com/science/article/pii/S0012365X09000776. Generalisations of de Bruijn Cycles and

Gray Codes/Graph Asymmetries/Hamiltonicity Problem for Vertex-Transitive (Cayley) Graphs.

[130] Thomas Fenz, Klaus-Tycho Foerster, Stefan Schmid, and Anaïs Villedieu. Efficient non-segregated routing for reconfigurable demand-aware networks. In *Proc. of IFIP Networking Conf.*, pages 1–9. IEEE Press, 2019.

[131] Piotr Berman and Marek Karpinski. Approximation hardness of bounded degree MIN-CSP and MIN-BISECTION. In *Proc. of 29th Int Colloq. on Automata, Languages and Programming (ICALP)*, pages 623–632, 2002.

[132] L.H. Clark and R.C. Entringer. The bisection width of cubic graphs. *Bull. Austral. Math. Soc.*, pages 389–396, 1988.

[133] Teofilo F. Gonzalez. Clustering to minimize the maximum intercluster distance. *Theoret. Comput. Sci.*, pages 293–306, 1985.

[134] OEIS Foundation Inc. Entry A000983 in the On-Line Encyclopedia of Integer Sequences, 2024. Published electronically at https://oeis.org/A000983.

[135] Brent N. Clark, Charles J. Colbourn, and David S. Johnson. Unit disk graphs. *Discrete Math.*, 86(1-3):165–177, 1990. ISSN 0012-365X. doi: 10.1016/0012-365X(90)90358-O.

[136] Siddharth Gupta, Guy Sa'ar, and Meirav Zehavi. Grid recognition: Classical and parameterized computational perspectives. *Journal of Computer and System Sciences*, 136:17–62, 2023. ISSN 0022-0000. doi: https://doi.org/10.1016/j.jcss.2023.02.008. URL https://www.sciencedirect.com/science/article/pii/S0022000023000259.

[137] Harvey L Abbott and M Katchalski. On the construction of snake in the box codes. *Utilitas Mathematica*, 40:97–116, 1991.

[138] Harvey L Abbott and Meir Katchalski. On the snake in the box problem. *Journal of Combinatorial Theory, Series B*, 45(1):13–24, 1988.

[139] Tala Eagling-Vose. Personal Communication, 2025.

[140] Guy E. Blelloch, Jeremy T. Fineman, and Julian Shun. Greedy sequential maximal independent set and matching are parallel on average. In *SPAA '12: Proceedings of the twenty-fourth annual ACM symposium on Parallelism in algorithms and architectures*, pages 308–317, June 2012.

[141] S. A. Kauffman. Metabolic stability and epigenesis in randomly connected nets. *Journal of Theoretical Biology*, 22:437–467, 1969.

[142] R. Thomas. Boolean formalization of genetic control circuits. *Journal of Theoretical Biology*, 42(3):563 – 585, 1973. ISSN 0022-5193. doi: 10.1016/0022-5193(73)90247-6.

[143] S. Bornholdt. Boolean network models of cellular regulation: prospects and limitations. *Journal of The Royal Society Interface*, 5(Suppl 1):S85–S94, 2008.

[144] J. Aracena, A. Richard, and L. Salinas. Synchronizing boolean networks asynchronously. *Journal of Computer and System Sciences*, 136:249–279, 2023.

[145] F. Robert. Iterations sur des ensembles finis et automates cellulaires contractants. *Linear Algebra and its Applications*, 29:393–412, 1980.

[146] Eric Goles. Dynamics of positive automata networks. *Theoretical Computer Science*, 41:19–32, 1985.

[147] E. Goles and M. Noual. Disjunctive networks and update schedules. *Advances in Applied Mathematics*, 48(5):646–662, 2012.

[148] Mathilde Noual and Sylvain Sené. Synchronism versus asynchronism in monotonic boolean automata networks. *Natural Computing*, 17:393–402, June 2018. ISSN 1572-9796. doi: 10.1007/s11047-016-9608-8. URL https://doi.org/10.1007/s11047-016-9608-8.

[149] Julio Aracena, Maximilien Gadouleau, Adrien Richard, and Lilian Salinas. Fixing monotone boolean networks asynchronously. *Information and Computation*, 274(104540), October 2020.

[150] Béla Bollobás and Imre Leader. Connectivity and dynamics for random subgraphs of the directed cube. *Israel Journal of Mathematics*, 83:321–328, 1993.

[151] Maximilien Gadouleau and Adrien Richard. On fixable families of boolean networks. In *Proc. Workshop on Asynchronous Cellular Automata*, pages 396–405, September 2018.

[152] A. Richard and P. Ruet. From kernels in directed graphs to fixed points and negative cycles in boolean networks. *Discrete Applied Mathematics*, 161(7):1106–1117, 2013.

[153] J. Aracena, A. Richard, and L. Salinas. Maximum number of fixed points in and-or-not networks. *Journal of Computer and System Sciences*, 80(7):1175 – 1190, 2014. ISSN 0022-0000. doi: http://dx.doi.org/10.1016/j.jcss.2014.04.025. URL http://www.sciencedirect.com/science/article/pii/S0022000014000695.

[154] Mohsen Ghaffari. Distributed maximal independent set using small messages. In *Proceedings of the 2019 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 805–820, 2019. doi: 10.1137/1.9781611975482.50. URL https://epubs.siam.org/doi/abs/10.1137/1.9781611975482.50.

[155] Yehuda Afek, Noga Alon, Ziv Bar-Joseph, Alejandro Cornejo, Bernhard Haeupler, and Fabian Kuhn. Beeping a maximal independent set. *Distributed computing*, 26(4):195–208, 2013.

[156] Artur Czumaj and Peter Davies. Communicating with beeps. *J. Parallel Distrib. Comput.*, 130(C):98–109, aug 2019. ISSN 0743-7315. doi: 10.1016/j.jpdc.2019.03.020. URL https://doi.org/10.1016/j.jpdc.2019.03.020.

[157] A. Casteigts, Y. Métivier, J.M. Robson, and A. Zemmari. Design patterns in beeping algorithms: Examples, emulation, and analysis. *Information and Computation*, 264:32–51, 2019. ISSN 0890-5401. doi: https://doi.org/10.1016/j.ic.2018.10.001. URL https://www.sciencedirect.com/science/article/pii/S0890540118301494.

[158] Edsger W. Dijkstra. Self-stabilizing systems in spite of distributed control. *Commun. ACM*, 17(11):643–644, nov 1974. ISSN 0001-0782. doi: 10.1145/361179.361202. URL https://doi.org/10.1145/361179.361202.

[159] Jukka Suomela. Survey of local algorithms. *ACM Comput. Surv.*, 45(2), mar 2013. ISSN 0360-0300. doi: 10.1145/2431211.2431223. URL https://doi.org/10.1145/2431211.2431223.

[160] Christoph Lenzen, Jukka Suomela, and Roger Wattenhofer. Local Algorithms: Self-Stabilization on Speed. In Sándor Fekete, Stefan Fischer, Martin Riedmiller, and Suri Subhash, editors, *Algorithmic Methods for Distributed Cooperative Systems*, volume 9371 of *Dagstuhl Seminar Proceedings (DagSemProc)*, pages 1–18, Dagstuhl, Germany, 2010. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi: 10.4230/DagSemProc.09371.3. URL https://drops.dagstuhl.de/entities/document/10.4230/DagSemProc.09371.3.

[161] Katrin Casel, Henning Fernau, Mehdi Khosravian Ghadikolaei, Jérôme Monnot, and Florian Sikora. On the complexity of solution extension of optimization problems. *Theoretical Computer Science*, 904:48–65, 2022. ISSN 0304-3975. URL https://www.sciencedirect.com/science/article/pii/S0304397521006253.

[162] Julio Aracena, Eric Goles, A. Moreira, and Lilian Salinas. On the robustness of update schedules in boolean networks. *BioSystems*, 97:1–8, 2009.

[163] Brendan D. McKay and Adolfo Piperno. Practical graph isomorphism, ii. *Journal of Symbolic Computation*, 60:94–112, 2014. ISSN 0747-7171. URL https://www.sciencedirect.com/science/article/pii/S0747717113001193.

[164] Ross M. McConnell and Jeremy Spinrad. Linear-time transitive orientation. In *Proc. 8th ACM-SIAM Symposium on Discrete Algorithms*, pages 19–25, 1997.

[165] Deutsche Bahn. Punctuality | Deutsche Bahn Interim Report 2024. https://zbir.deutschebahn.com/2024/en/interim-group-management-report-unaudited/product-quality-and-digitalization/punctuality/, 2024. [Accessed 19-09-2024].

[166] Hendrik Molter, Malte Renken, and Philipp Zschoche. Temporal reachability minimization: Delaying vs. deleting. *Journal of Computer and System Sciences*, 144:103549, September 2024. ISSN 00220000. doi: 10.1016/j.jcss.2024.103549. URL https://linkinghub.elsevier.com/retrieve/pii/S0022000024000448.

[167] Eugen Füchsle, Hendrik Molter, Rolf Niedermeier, and Malte Renken. Temporal Connectivity: Coping with Foreseen and Unforeseen Delays, January 2022. URL http://arxiv.org/abs/2201.05011. arXiv:2201.05011 [cs].

[168] Anita Schöbel. A Model for the Delay Management Problem based on Mixed-Integer-Programming. *Electronic Notes in Theoretical Computer Science*, 50(1):1–10, August 2001. ISSN 1571-0661. doi: 10.1016/S1571-0661(04)00160-4. URL https://www.sciencedirect.com/science/article/pii/S1571066104001604.

[169] Twan Dollevoet, Dennis Huisman, Marie Schmidt, and Anita Schöbel. Delay Management with Rerouting of Passengers. *Transportation Science*, 46(1):74–89, February 2012. ISSN 0041-1655, 1526-5447. doi: 10.1287/trsc.1110.0375. URL https://pubsonline.informs.org/doi/10.1287/trsc.1110.0375.

[170] Stefan Binder, Yousef Maknoon, and Michel Bierlaire. The multi-objective railway timetable rescheduling problem. *Transportation Research Part C: Emerging Technologies*, 78:78–94, May 2017. ISSN 0968-090X. doi: 10.1016/j.trc.2017.02.001. URL https://www.sciencedirect.com/science/article/pii/S0968090X17300414.

[171] Andreas Ginkel and Anita Schöbel. To Wait or Not to Wait? The Bicriteria Delay Management Problem in Public Transportation. *Transportation Science*, 41(4):527–538, November 2007. ISSN 0041-1655, 1526-5447. doi: 10.1287/trsc.1070.0212. URL https://pubsonline.informs.org/doi/10.1287/trsc.1070.0212.

[172] Géraldine Heilporn, Luigi De Giovanni, and Martine Labbé. Optimization models for the single delay management problem in public transportation. *European Journal of Operational Research*, 189(3):762–774, September 2008. ISSN 03772217. doi: 10.1016/j.ejor.2006.10.065. URL https://linkinghub.elsevier.com/retrieve/pii/S0377221706011830.

[173] Lucas P. Veelenturf, Martin P. Kidd, Valentina Cacchiani, Leo G. Kroon, and Paolo Toth. A Railway Timetable Rescheduling Approach for Handling Large-Scale Disruptions. *Transportation Science*, 50(3):841–862, August 2016. ISSN 0041-1655, 1526-5447. doi: 10.1287/trsc.2015.0618. URL https://pubsonline.informs.org/doi/10.1287/trsc.2015.0618.

[174] Yongqiu Zhu and Rob M. P. Goverde. Integrated timetable rescheduling and passenger reassignment during railway disruptions. *Transportation Research*

*Part B: Methodological*, 140:282–314, October 2020. ISSN 0191-2615. doi: 10.1016/j.trb.2020.09.001. URL https://www.sciencedirect.com/science/article/pii/S0191261520303878.

[175] Michael Gatto, Björn Glaus, Riko Jacob, Leon Peeters, and Peter Widmayer. Railway Delay Management: Exploring Its Algorithmic Complexity. In *Algorithm Theory - SWAT 2004*, volume 3111, pages 199–211. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004. ISBN 978-3-540-22339-9 978-3-540-27810-8. doi: 10.1007/978-3-540-27810-8_18. URL http://link.springer.com/10.1007/978-3-540-27810-8_18. Series Title: Lecture Notes in Computer Science.

[176] Michael Gatto, Riko Jacob, Leon Peeters, and Anita Schöbel. The Computational Complexity of Delay Management. In *Graph-Theoretic Concepts in Computer Science*, volume 3787, pages 227–238. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005. ISBN 978-3-540-31000-6 978-3-540-31468-4. doi: 10.1007/11604686_20. URL http://link.springer.com/10.1007/11604686_20. Series Title: Lecture Notes in Computer Science.

[177] Michael Schachtebeck. *Delay Management in Public Transportation: Capacities, Robustness, and Integration*. PhD thesis, Georg-August-University Göttingen, 2010. URL https://ediss.uni-goettingen.de/handle/11858/00-1735-0000-0006-B3CE-4.

[178] S. Carosi, S. Gualandi, F. Malucelli, and E. Tresoldi. Delay Management in Public Transportation: Service Regularity Issues and Crew Re-scheduling. *Transportation Research Procedia*, 10:483–492, 2015. ISSN 23521465. doi: 10.1016/j.trpro.2015.09.002. URL https://linkinghub.elsevier.com/retrieve/pii/S2352146515001891.

[179] Federico Malucelli and Emanuele Tresoldi. Delay and disruption management in local public transportation via real-time vehicle and crew re-scheduling: a case study. *Public Transport*, 11(1):1–25, June 2019. ISSN 1866-749X, 1613-7159. doi: 10.1007/s12469-019-00196-y. URL http://link.springer.com/10.1007/s12469-019-00196-y.

[180] Chuntian Zhang, Yuan Gao, Valentina Cacchiani, Lixing Yang, and Ziyou Gao. Train rescheduling for large-scale disruptions in a large-scale railway network. *Transportation Research Part B: Methodological*, 174:102786, August 2023. ISSN 0191-2615. doi: 10.1016/j.trb.2023.102786. URL https://www.sciencedirect.com/science/article/pii/S019126152300111X.

[181] Geoffrey Scozzaro, Clara Buire, Daniel Delahaye, and Aude Marzuoli. Optimizing air-rail travel connections: A data-driven delay management strategy for seamless passenger journeys. In *SESAR Innovation Days*, 2023.

[182] Valentina Cacchiani, Dennis Huisman, Martin Kidd, Leo Kroon, Paolo Toth, Lucas Veelenturf, and Joris Wagenaar. An overview of recovery models and algorithms for real-time railway rescheduling. *Transportation Research Part B: Methodological*, 63:15–37, May 2014. ISSN 01912615. doi: 10.1016/j.trb.2014.01.009. URL https://linkinghub.elsevier.com/retrieve/pii/S0191261514000198.

[183] Eva König. A review on railway delay management. *Public Transport*, 12(2):335–361, June 2020. ISSN 1866-749X, 1613-7159. doi: 10.1007/s12469-020-00233-1. URL http://link.springer.com/10.1007/s12469-020-00233-1.

[184] Schöbel, Anita. *Optimization in Public Transportation*, volume 3 of *Springer Optimization and Its Applications*. Springer US, Boston, MA, 2006. ISBN 978-0-387-32896-6. doi: 10.1007/978-0-387-36643-2. URL http://link.springer.com/10.1007/978-0-387-36643-2.

[185] Leo G. Kroon, Rommert Dekker, and Michiel J. C. M. Vromans. Cyclic Railway Timetabling: A Stochastic Optimization Approach. In Frank Geraets, Leo Kroon, Anita Schoebel, Dorothea Wagner, and Christos D. Zaroliagis, editors, *Algorithmic Methods for Railway Optimization*, volume 4359, pages 41–66. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007. ISBN 978-3-540-74245-6. doi: 10.1007/978-3-540-74247-0_2. URL http://link.springer.com/10.1007/978-3-540-74247-0_2. Series Title: Lecture Notes in Computer Science.

[186] Huanhuan Wu, James Cheng, Silu Huang, Yiping Ke, Yi Lu, and Yanyan Xu. Path problems in temporal graphs. *Proc. VLDB Endow.*, 7(9):721–732, may 2014. ISSN 2150-8097. doi: 10.14778/2732939.2732945. URL https://doi.org/10.14778/2732939.2732945.

[187] N. Karmarkar. A new polynomial-time algorithm for linear programming. In *Proceedings of the Sixteenth Annual ACM Symposium on Theory of Computing*, STOC '84, page 302–311, New York, NY, USA, 1984. Association for Computing Machinery. ISBN 0897911334. doi: 10.1145/800057.808695. URL https://doi.org/10.1145/800057.808695.

[188] A. B. Kahn. Topological sorting of large networks. *Communications of the ACM*, 5(11):558–562, November 1962. ISSN 0001-0782, 1557-7317. doi: 10.1145/368996.369025. URL https://dl.acm.org/doi/10.1145/368996.369025.

[189] Ivan Tadeu Ferreira Antunes Filho. *Characterizing Boolean satisfiability variants*. PhD thesis, Massachusetts Institute of Technology, 2019.

[190] Dániel Marx. NP-completeness of list coloring and precoloring extension on the edges of planar graphs. *Journal of Graph Theory*, 49(4):313–324, 2005. ISSN 1097-0118. doi: 10.1002/jgt.20085. URL

https://onlinelibrary.wiley.com/doi/abs/10.1002/jgt.20085. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/jgt.20085.

[191] B. M. E. Moret. Planar nae3sat is in p. *SIGACT News*, 19(2):51–54, June 1988. ISSN 0163-5700. doi: 10.1145/49097.49099. URL https://doi.org/10.1145/49097.49099.

[192] Madhura Dutta, Anil Maheshwari, and Subhas C. Nandy. Partial domination in some geometric intersection graphs. In Daya Gaur and Rogers Mathew, editors, *Algorithms and Discrete Applied Mathematics*, pages 121–133, Cham, 2025. Springer Nature Switzerland. ISBN 978-3-031-83438-7.

[193] Benjamin M. Case, Stephen T. Hedetniemi, Renu C. Laskar, and Drew J. Lipman. Partial domination in graphs, 2017. URL https://arxiv.org/abs/1705.03096.

[194] Csilla Bujtás, Michael A. Henning, and Sandi Klavžar. Partial domination in supercubic graphs, 2023. URL https://arxiv.org/abs/2305.19820.

[195] David C. Kutner and Iain A. Stewart. Reconfigurable routing in data center networks, 2024. URL https://arxiv.org/abs/2401.13359.

[196] Miroslav Chlebík and Janka Chlebíková. The complexity of combinatorial optimization problems on d-dimensional boxes. *SIAM Journal on Discrete Mathematics*, 21(1):158–169, 2007. doi: 10.1137/050629276. URL https://doi.org/10.1137/050629276.

[197] Glencora Borradaile and Hung Le. Optimal Dynamic Program for r-Domination Problems over Tree Decompositions. In Jiong Guo and Danny Hermelin, editors, *11th International Symposium on Parameterized and Exact Computation (IPEC 2016)*, volume 63 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 8:1–8:23, Dagstuhl, Germany, 2017. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. ISBN 978-3-95977-023-1. doi: 10.4230/LIPIcs.IPEC.2016.8. URL https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.IPEC.2016.8.

[198] Stephen R. Mahaney. Sparse complete sets for np: Solution of a conjecture of berman and hartmanis. *Journal of Computer and System Sciences*, 25(2):130–143, 1982. ISSN 0022-0000. doi: https://doi.org/10.1016/0022-0000(82)90002-2. URL https://www.sciencedirect.com/science/article/pii/0022000082900022.

[199] Subhash Khot and Oded Regev. Vertex cover might be hard to approximate to within 2-$\epsilon$. *Journal of Computer and System Sciences*, 74(3):335–349, 2008. ISSN 0022-0000. doi: https://doi.org/10.1016/j.jcss.2007.06.019. URL https://www.sciencedirect.com/science/article/pii/S0022000007000864. Computational Complexity 2003.

[200] Irit Dinur, Venkatesan Guruswami, Subhash Khot, and Oded Regev. A new multilayered pcp and the hardness of hypergraph vertex cover. In *Proceedings of the Thirty-Fifth Annual ACM Symposium on Theory of Computing*, STOC '03, page 595–601, New York, NY, USA, 2003. Association for Computing Machinery. ISBN 1581136749. doi: 10.1145/780542.780629. URL https://doi.org/10.1145/780542.780629.

[201] Viggo Kann. Maximum bounded 3-dimensional matching is max snp-complete. *Information Processing Letters*, 37(1):27–35, 1991. ISSN 0020-0190. doi: https://doi.org/10.1016/0020-0190(91)90246-E. URL https://www.sciencedirect.com/science/article/pii/002001909190246E.

[202] Hongliang Lu, Xingxing Yu, and Xiaofan Yuan. Nearly perfect matchings in uniform hypergraphs. *SIAM Journal on Discrete Mathematics*, 35(2):1022–1049, 2021. doi: 10.1137/19M1300662. URL https://doi.org/10.1137/19M1300662.

[203] Tatjana Schmidt. *Computational complexity of SAT, XSAT and NAE-SAT for linear and mixed Horn CNF formulas.* PhD thesis, Universität zu Köln, 2010.

[204] Alan Frieze and Dhruv Mubayi. Coloring simple hypergraphs. *Journal of Combinatorial Theory, Series B*, 103(6):767–794, 2013. ISSN 0095-8956. doi: https://doi.org/10.1016/j.jctb.2013.09.003. URL https://www.sciencedirect.com/science/article/pii/S0095895613000658.

[205] David Ellis and Nathan Linial. On regular hypergraphs of high girth. *arXiv preprint arXiv:1302.5090*, 2013.