University of Bremen

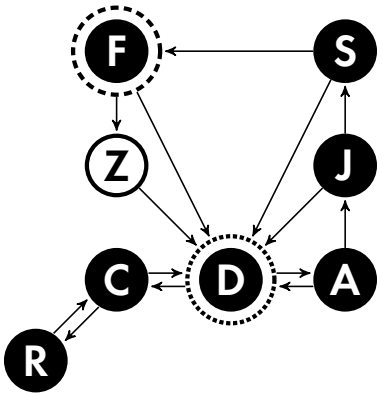
Bachelor Thesis

Bi-Kelly-Width

A New Width Measure for Directed Graphs

Enna Gerhard

May 14, 2021



Supervision and first reviewer
Prof. Dr. Sebastian Siebertz

Second reviewer
Dr. Sabine Kuske

Abstract. The notion of *Tree-Width* is a width measure for undirected graphs with a wealth of combinatorial and algorithmic applications. Therefore, many attempts have been made to adapt Tree-Width for directed graphs, leading for example, to the concepts of *Directed Tree-Width*, *DAG-Width*, and *Kelly-Width*. All of these measures for directed graphs however do not lead to a satisfying algorithmic theory. One reason is that all acyclic graphs have width zero or one with respect to all the measures listed above, while for example the directed edge disjoint paths problem is unlikely to be solved in polynomial time on all acyclic graphs. Hence, these measures fall short of distinguishing between »simple« and »complex« directed graphs.

We introduce a new width measure for directed graphs that aims to address this shortcoming of the traditional width measures. Our measure, named *Bi-Kelly-Width*, is inspired by Kelly-Width, which is defined in terms of elimination orders. The quality of an elimination order for Bi-Kelly-Width is measured by forward and backward connectivity properties, which in particular results in a separation of dense and sparse acyclic graphs. We develop a structure theory of digraphs of bounded Bi-Kelly-Width by providing characterizations in terms of cops and robber games, elimination orders, fill-in graphs and partial bi-directed k-trees. Our results show an interesting new measure sandwiched between undirected Tree-Width and Kelly-Width and open the door for new algorithmic developments.

Kurzfassung. Das Konzept der Baumweite (Tree-Width) ist ein Weitemaß für ungerichtete Graphen. Es gibt eine Vielzahl kombinatorischer und algorithmischer Anwendungen. In der Literatur finden sich viele Versuche, ein Äquivalent zur Baumweite auf gerichteten Graphen zu finden. Hierzu gehören die *Gerichtete Baumweite* (Directed Tree-Width), *Gerichtete-Kreisfreiheits-Weite* (DAG-Width) und *Kelly-Weite* (Kelly-Width). Für einige Anwendungen sind diese jedoch allesamt nicht zufriedenstellend. Einer der Gründe ist, dass all diese Weitemaße auf kreisfreien Graphen einen Wert von Null annehmen, während Probleme wie das Überschneidungsfreie-Wege-Problem (Directed Edge-Disjoint Paths Problem) für gerichtete Graphen auch bei Kreisfreiheit wahrscheinlich nicht in polynomieller Zeit lösbar sind. Aus diesem Grund sind sie nicht dazu in der Lage, in dieser Frage zwischen »einfachen« und »komplexen« gerichteten Graphen zu unterscheiden.

Wir stellen ein neues Weitemaß für gerichtete Graphen vor, das versucht, diese Schwierigkeiten zu lösen. Bezeichnen werden wir es als *Bi-Kelly-Weite*, in Anlehnung an die ebenfalls über Eliminierungsreihenfolgen definierte Kelly-Weite. Die Weite einer Eliminierungsreihenfolge wird durch den Zusammenhang innerhalb des Graphen bestimmt, wobei auch auf Rückrichtungen von Pfaden geachtet wird. Dies ermöglicht auch die Unterscheidung von dichten und dünn besetzten kreisfreien Graphen. Zusätzlich entwickeln wir eine umfangreiche Strukturtheorie der gerichteten Graphen mit beschränkter Bi-Kelly-Weite, indem wir verschiedene äquivalente Beschreibungen betrachten: Räuber- und Gendarmspiele, Eliminierungsreihenfolgen, Auffüllgraphen und partielle beidseitig gerichtete k-Bäume. Unser Ergebnis ist ein interessantes neues Weitemaß. Es liegt zwischen ungerichteter Baumweite und Kelly-Weite und könnte die Grundlage für zukünftige algorithmische Anwendungen sein.

Contents

1. Introduction

1.1. Motivation

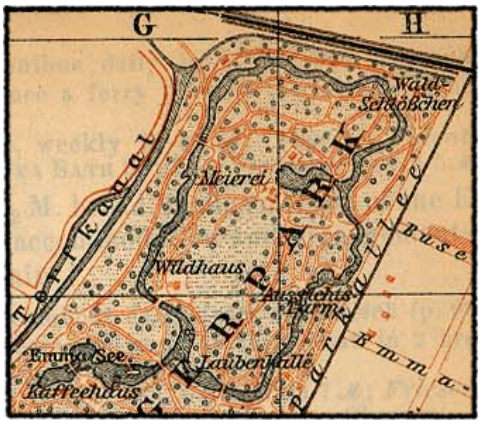
Our world provides us with ever more complex problems to solve. Large parts of our economical, physical and social lives are relying on digital infrastructure. We try to use resources more and more efficiently, for example to reduce waste and overproduction. Computer science needs to make a strong contribution towards several of these goals (Schneidewind et al., 2018). It offers tools to formalize real-world problems and methods to solve them efficiently.

Many of the current areas of innovation are focused around networks and connectivity. For example, a zero waste economy relies on a complete reuse of resources and tracing materials until their end of life. In the future, our power supply might be organized in smart grids instead of centralized huge power plants and top down distribution of power. Hybrid routing paradigms that incorporate public transport and autonomous driving are likely to make commuting more efficient.

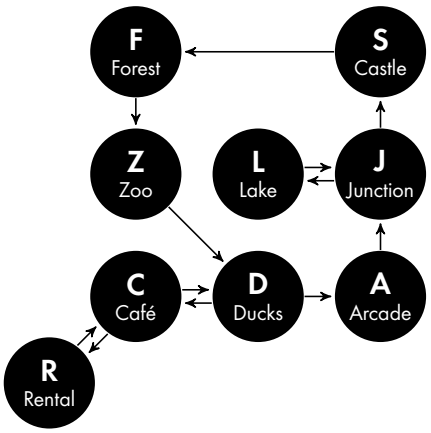
Tackling these problems with algorithms requires creating and understanding a digital representation of our world. Many of these problems can conveniently be formalized as network and graph problems. A graph uses »vertices« as abstract representations of objects and concepts out of our world and »edges« between the vertices to represent relations, connections or interactions between the concepts represented by the vertices.

Directed Graph	**Vertex**	**Edge**
Academia	Publication	Citation
Game	Board layout	Legal move
Infectious disease	Person	Possible infection
Program	Code block	Jump
Scheduling	Task	Constraint
Social network	User	Message
Transportation	Location	Way of passage
Wildlife	Species	Predator-prey relation
World Wide Web	Website	Hyperlink

Figure 1.1.: A selection of concepts that naturally translate to directed graphs (after Sedgewick and Wayne, 2007)



(a) Map (excerpt from Baedeker, 1910)



(b) Graph representation

Figure 1.2.: The boat course in the Bürgerpark in Bremen, Germany

Let us take a look at a simple example, Figure 1.2. Both the map and the graph contain a representation of the boat course through the Bürgerpark in Bremen. However, they have different primary purposes. The map contains a large amount of information, with the purpose of informing visitors, but it is difficult to work with it using algorithms. The graph focuses on nine interesting points on its course and represents these as vertices. The round vertices are connected by edges, depicted as arrows. They also indicate a direction or a connection in both ways as seen between the vertex labeled Ducks and the vertex labeled Café. The underlying abstract representation is directly accessible by algorithms.

Problems that we might want to solve for our boat course could include planning of boat trips which respect one way passages. You could leave the lake aside on a round trip or decide to go on another loop when getting back to the Island of Ducks. We could also define a maximum amount of boats allowed at a single site, for example only two on the rental while seven would be acceptable for the arcade. Then, we could also take movements into account. While it would still be possible to manage this rather simple boat course by hand, we would quickly need to resort to finding an algorithm that allows a computer to solve this task for larger graphs and boat courses, respectively.

Another example of possible real-world applications is the Bremen tram rail map, Figure 1.3 on the next page. It represents several rails and the way these are connected at junctions. Problems that are closely connected to railway maps are routing, scheduling and capacity problems that are restricted by the underlying rail structure.

Concerning routing problems, let us try to find a round trip connecting all points of interest for our boat course, starting and ending in the rental, ideally only visiting each point once. In graph theory, this is known as the Directed Hamilton Cycle Problem. A naive approach to solve this problem is to try out all possible sequences of vertices and test if they form a directed cycle. For this boat course, this would be $9 \cdot 8 \cdot \ldots \cdot 2 \cdot 1 = 9! = 362880$ – no problem for a computer.

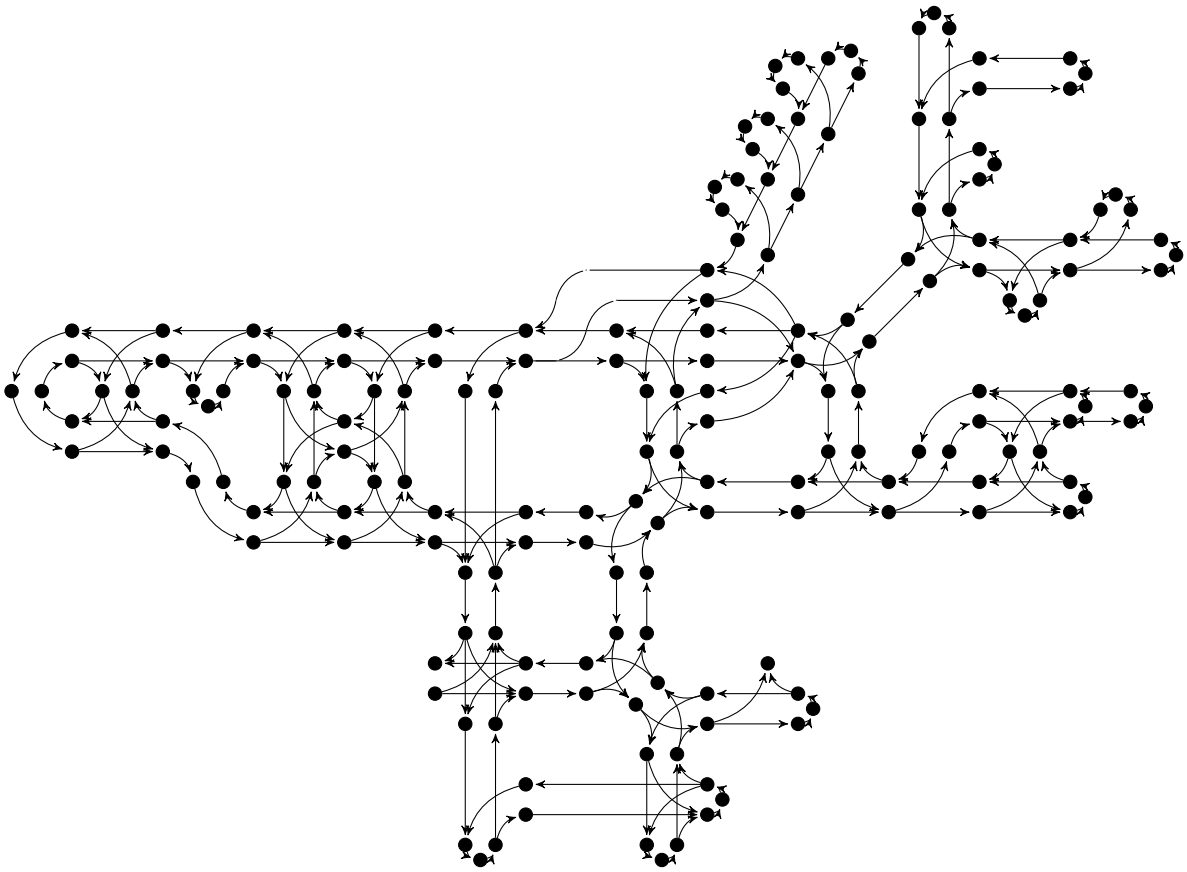


Figure 1.3.: Bremen tram rail map

The rail map on the other hand uses over one hundred vertices, a magnitude larger than nine from the boat course. It is obvious that solving non-trivial tasks by hand has become impossible. However, for large graphs, even a computer will run into difficulties. If we multiply $100 \cdot 99 \cdot \ldots \cdot 2 \cdot 1 = 100! \approx 10^{158}$, we end up with an amount of required operations that are not feasible even for a modern computer. A current multi-core computer could have made around $13.82\text{bn years} \cdot 20 \cdot 3\text{GHz} \approx 10^{31}$ calculations if it started at the beginning of the universe.

The Directed Hamilton Cycle Problem is just one problem that is NP-*complete*: Verifying an existing solution is easy – we can take a suggested round trip and verify that each connection is permissible – and only takes a relatively small amount of steps. For problems that are in NP, this is possible in polynomial time. We can find a polynomial $an^b + c$ such that we can verify a solution in this amount of steps only depending on the number of vertices n in our graph. However, we do not know exact algorithms that are efficient for general instances. When looking at the worst case running time for existing algorithms, we end up with the same running time as with a naive approach to try out every possible combination, ignoring constant factors.

The Directed Edge Disjoint Paths Problem is another key problem. Given a graph and k pairs of vertices (s_i, t_i) with $i \in \{1, \ldots, k\}$, the Directed Edge Disjoint Paths Problem is to determine whether we can find paths that are not allowed to share any edges to connect all starting points s_i with their corresponding terminals t_i. It is known to be NP-complete even on acyclic digraphs. As such, it will be a major motivation for this thesis.

1.1.1. Adding parameters to complexity

When creating algorithms we can find at least some solutions to handle larger input sizes. Parallelization is in a way trying to reduce required time by performing multiple independent computations at once. But it can only divide the required time by a constant factor of processors used, which still does not result in a feasible solution. Dividing 10^{158} by one billion already requires one billion computers but only leads to 10^{149}.

On the other hand, if a perfect result is not as important, for example finding the fastest tram versus one that arrives 30 seconds later, an approximation can be used. This will sacrifice an exact answer for faster computation. We will usually retain an approximation guarantee, for example the optimal solution being at most twice as good as the approximation. A second option is to use heuristics, which may improve running times, though without any approximation guarantees and often without finding an exact solution either.

Lastly, our algorithm may not need to be able to solve all instances of a size as fast as others. It may be possible to add additional parameters that describe the complexity of a problem. Depending on these parameters, we still get an exact prediction about the worst case for required running time. For small values of these parameters, running time becomes fast, while for large parameters it may be slow. Such a parameter allows for a fast, exact solution to a problem, but sacrifices keeping this promise for instances where the parameters are large.

Depicting these trade-offs, we obtain a triangle (Figure 1.4) where we are only allowed to pick two corners. This works similar to the project management triangle (time, cost, quality) or the cleaning triangle (time, force, chemicals).

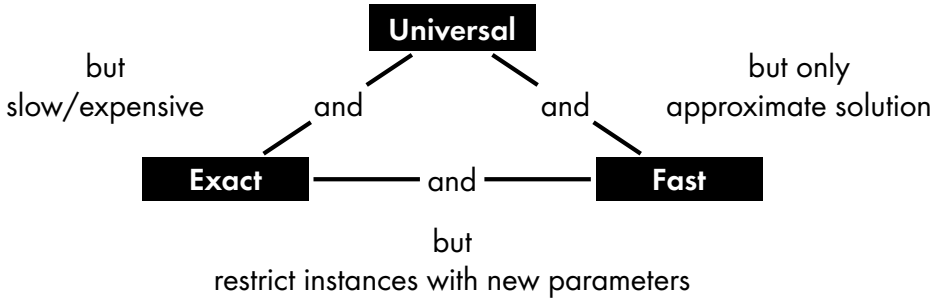


Figure 1.4.: Conflicting properties of algorithms

Adding additional parameters into the analysis of complexity in algorithms allows for a fine-grained view on complexity. Classical complexity theory usually only relies on one input size. For graph problems, this tends to be the raw number of edges and vertices in a graph, n. The parameterized approach however allows distinguishing instances on a completely different basis, taking also structural measures of the input instances into account (Cygan et al., 2016).

In this thesis we aim to develop a new structural parameter for directed graphs. We want to be able to describe on what instances we can hope for exact and fast algorithms.

1.1.2. Structural complexity of graphs

In graph theory, more efficient algorithms can be created if the underlying graph is well-structured. The main goal of this thesis is to understand these structures in directed graphs and to explore a new width measure for directed graphs. The ultimate goal of such a structure theory is to develop complexity measures that can distinguish exactly between simple and hard instances of problems and use this for creating efficient algorithms on those simple instances.

A graph property on undirected graphs that has been researched in great detail is Tree-Width. Works are dating back as far as the seventies (Bertelé and Brioschi, 1972; Halin, 1976). The principle behind the current definition was introduced by Robertson and Seymour in 1984. Since then, many equivalent definitions were found. At this point we will only look at one particularly intuitive description of Tree-Width, as we will explain it in detail in Section 2.2 on page 18. Seymour and Thomas (1993) used a cops and robber game on graphs. In this game a team of cops is chasing a robber, and the number of cops required for a winning strategy against a single robber depends on the Tree-Width and vice-versa. This is similar to other searching techniques introduced earlier (Kirousis and Papadimitriou, 1986; LaPaugh, 1993).

There are many efficient algorithms taking trees as an input. Searching trees and calculating properties on trees can generally be done efficiently. Intuitively, this will also apply to a graph that is similar to a tree, and this similarity with a tree is captured by the parameter Tree-Width. We could, for example, track properties that apply to an already searched part of our graph using dynamic programming (Bodlaender, 1988), just as we might have calculated an intermediate result based on the branch of a tree. In effect, if we use Tree-Width as a fixed parameter, we are able to solve many problems that are NP-complete in general in polynomial or even linear time. For example, the Hamilton Cycle Problem can be solved in time $f(t) \cdot n$, where t is the Tree-Width of the input graph, n is the number of vertices of the input graph and f is some computable function depending only on the Tree-Width (Arnborg, Corneil, et al., 1987). We call this fixed-parameter tractability (FPT). The superpolynomial dependence on some parameter can very likely not be avoided, as the Hamilton Cycle Problem is NP-complete, but the dependence is only on the Tree-Width and not on the size of the graph.

Many other applications for Tree-Width have been found (Bodlaender, 1993). Arnborg and Proskurowski (1989) achieved several results for solving generally NP-hard problems such as Independent Set, Dominating Set, Graph K-Colorability and Network Reliability on instances with bounded Tree-Width. Courcelle (1990) proved that graph properties defined in monadic second-order logic (MSO) can be determined in linear time, which is a very powerful tool. It helps both with solving decision problems and optimisation problems on graphs (Courcelle and Engelfriet, 2012). Furthermore, the undirected Edge-Disjoint Paths Problem can be solved in time $f(k,t) \cdot n^{O(1)}$, where k is the parameter of the problem, t is the undirected Tree-Width of the input graph with n vertices and $O(1)$ is a constant (Robertson and Seymour, 1995). The function f is large, however, for small values of k and t we indeed get fast running times. Other applications include circuit design (Möhring, 1990), or evolution theory with solving the Perfect Phylogeny problem (Bodlaender and Kloks, 1992).

1.1.3. Structural complexity of directed graphs

As we have seen in the initial examples, many important problems can be naturally formalized as directed graphs. Tree-Width on the other hand only applies to undirected graphs. Sure, one could remove the directions of edges and treat a directed graph as an undirected one, but this is not satisfying. A lot of valuable information can get lost during that process (see Figure 2.3 on page 17).

Therefore, there have been several attempts to translate the notion of Tree-Width to directed graphs. These include the notions of Directed Tree-Width (Johnson et al., 2001; Reed, 1999) and DAG-Width (Berwanger et al., 2006; Obdržálek, 2006). Hunter and Kreutzer (2008) transferred the Tree-Width game with similar principles to directed graphs. Surprisingly however, the number of cops required does not correspond to Directed Tree-Width, but rather gives rise to a new width measure they called Kelly-Width. One motivation behind these directed width measures is that they shall distinguish between simple and complex directed graphs, taking edge directions into account. On the other hand, when the directed version is applied to symmetric graphs (that can be understood as undirected graphs), then the measure should be equal to undirected Tree-Width.

Unfortunately, there are some fundamental restrictions on what we can expect from directed width measures. For example, the model-checking problem for monadic second-order logic can be solved efficiently essentially only if the underlying undirected graphs have bounded Tree-Width (Makowsky and Mariño, 2003). This rules out very general algorithmic tractability results for directed width measures that generalize undirected Tree-Width.

However, several possible applications remain. For example, the Directed Edge Disjoint Paths Problem can be solved in time $n^{O(k+w)}$, where k is the number of paths to be found and w is the Directed Tree-Width of the input graph. We call this complexity class W[1].

The assumption FPT ≠ W[1] can be seen as a parameterized analog of the assumption P ≠ NP and are even related. The problems in FPT admit efficient parameterized algorithms, while we do not expect that such algorithms exist for W[1]-hard problems (without going further into the details of parameterized complexity theory, for more details see Downey and Fellows, 2013). As a result, many connectivity problems can be solved efficiently only on graphs of small Directed Tree-Width. This means that we cannot expect that there is an algorithm running in time $f(k) \cdot n^{g(w)}$ for any functions f, g, as Slivkins (2010) proved that the Directed Edge Disjoint Paths Problem is already W[1]-hard on acyclic graphs, which have Directed Tree-Width 0. A width measure that distinguishes between simple and difficult instances must hence distinguish between simple and difficult acyclic digraphs (Ganian, Langer, et al., 2014).

While Kelly-Width is never smaller than Directed Tree-Width, it still is not the refined measure that we are looking for (Ganian, Meister, et al., 2016). In particular, all acyclic digraphs have Kelly-Width 0, so an FPT algorithm with parameter k for the Directed Edge Disjoint Paths Problem characterized by Kelly-Width is still ruled out.

1.2. Objective of this thesis

In this thesis, we take the very intuitive definition of the cops and robber game and give additional power to the robber. In the game we allow the robber to run not only along directed paths, but also along directed paths in the reverse direction.

This typically requires a larger number of cops to catch the robber than in the game for Kelly-Width and leads to the new measure Bi-Kelly-Width. The new measure lies somewhere between undirected Tree-Width and Kelly-Width. On symmetric graphs, it collapses to undirected Tree-Width, as required by all well-behaved directed Tree-Width-like measures. Furthermore, it is not bounded on all acyclic digraphs, which will potentially allow for an FPT algorithm for the Directed Edge Disjoint Paths Problem. In particular, Bi-Kelly-Width is unbounded on the worst-case construction of Slivkins (2010).

While we were unable to solve this difficult algorithmic question, we develop a nice structure theory for the new width measure. In particular, we will provide various equivalent characterizations closely related to ones already known for Tree-Width, showing that we have found a robust new measure for directed graphs.

1.3. Outline

This thesis is divided into five chapters. You are currently reading Chapter 1, Introduction.

Next up, Chapter 2, Definitions and Concepts, will introduce both the formal definitions used throughout the thesis, and the most important existing research that we will build upon: Tree-Width in Section 2.2 and then Kelly-Width in Section 2.3.

After that, we can construct Bi-Kelly-Width itself in Chapter 3 on page 29. We will start with the very intuitive idea using cops and robber games, then focus on the more formal elimination orders that can be applied to them. Furthermore, we will look at partial bi-directed k-trees as a more constructive approach. All of these are identical, as we will prove in Section 3.4. After that, we can observe different properties of Bi-Kelly-Width.

All in all, we will look at nine width representations that are closely related to each other.

	Tree-Width	Kelly-Width	Bi-Kelly-Width
Game	Section 2.2.2	Section 2.3.1	Section 3.1
Elimination orders	Section 2.2.1*	Section 2.3.2	Section 3.2
Graph construction	Section 2.2.3	Section 2.3.4	Section 3.3

* Tree-decompositions differ from linear orders, but are closely related.

Table 1.1.: Different and equivalent reach representations

With this new theoretic toolset at hands, we will compare the similarities and difference between the three aforementioned width-measures in Chapter 4, Interesting Graphs. Finally, we will conclude in Chapter 5 with an analysis of the newly introduced width measure, and an outlook on possible further developments.

The Appendix on page 62 lists references used throughout this thesis and contains a list of figures. It also provides a short overview of definitions.

2. Definitions and Concepts

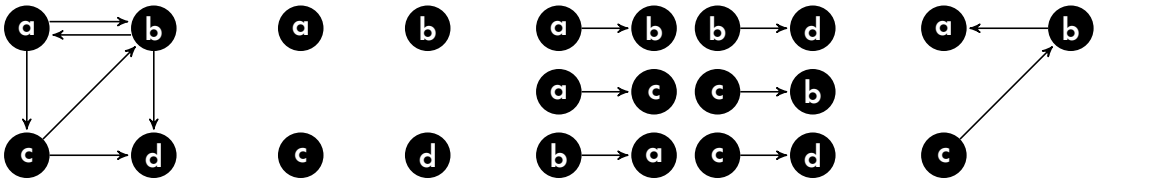
In the following, we will use standard notation graph theory (for example Bang-Jensen and Gutin, 2008) wherever possible. As this tends to be ambiguous in some cases and some definitions may not be known, we will go over the most important ones.

2.1. General graph notation and definitions

A *graph* $G = (V, E)$ consists of a set of *vertices* $V(G) = V$ and a set of *edges* $E(G) = E \subseteq V(G)^2$, hence every edge $e \in E(G)$ is a pair of vertices $e = (u, v)$ with $u, v \in V(G)$. Unless otherwise noted, graphs are directed, otherwise, their edges are undirected 2-element sets. Directed graphs are sometimes called digraphs. As we defined them using sets, we will simply ignore adding vertices or edges twice. We also do not allow loops (w, w). Connecting two vertices with edges in both directions is allowed.

Graphs, for example $G = (\{a, b, c, d\}, \{(a, b), (a, c), (b, a), (b, d), (c, b), (c, d)\})$ as depicted in Figure 2.1a, may be represented visually. Vertices (2.1b) are represented as circles. They are usually labeled and, in this thesis, generally filled black if they do not have a special meaning. Edges are represented as arrows between two vertices (see 2.1c). In this example, a and b are connected in both directions.

A *subgraph* H is a graph itself and is related to its parent graph G such that it contains a subset of its edges and vertices. We write $H \subseteq G$ with the conditions $V(H) \subseteq V(G)$ and $E(H) \subseteq E(G)$. A subset of vertices $V' \subseteq V(G)$ may be used to create an *induced subgraph*



(a) A graph G (b) Vertices $V(G)$ (c) Edges $E(G)$ (d) A subgraph $H \subseteq G$

Figure 2.1.: Basic graph examples

$G[V']$ with $V(G[V']) = V'$ and $E(G[V']) = \{(u, v) \in E(G) \mid u \in V' \wedge v \in V'\}$. Figure 2.1d displays an example where both a vertex and its incident edges, as well as two other edges, have been removed.

We may intersect graphs $G \cap H$ such that the result is the largest subgraph of both parents $G \cap H = (V(G) \cap V(H), E(G) \cap E(H))$ or join graphs $G \cup H$ analogously: $G \cup H = (V(G) \cup V(H), E(G) \cup E(H))$.

A *linear order* is a binary relation $\leq$ on a set X that is

- *antisymmetric*: if two elements are related in both directions, they are identical: $\forall x, y \in X : x \leq y \wedge y \leq x \Rightarrow x = y$,
- *transitive*: $\forall x, y, z \in X : x \leq y \wedge y \leq z \Rightarrow x \leq z$,
- and *connex*: any two elements are related in either way: $\forall x, y \in X : x \neq y \Rightarrow x \leq y \vee y \leq x$.

Sometimes, when X is finite, we will address the ordered elements of X simply as $(x_1, \ldots, x_n)$. In this case, $\leq$ will refer to the placement of elements in this order.

Having defined a linear order $\leq$ on $V(G)$, we can use $V[\leq v] = \{v_i \in V \mid v_i \leq v\}$ for the set of elements smaller than or equal to v. $G[\leq v]$ is the subgraph $G[V[\leq v]]$. $V[\geq v]$, $V[> v]$, $V[< v]$, $G[\geq v]$, $G[> v]$ and $G[< v]$ work as expected.

A *path* $u_1 \rightsquigarrow u_n$ is a sequence of vertices $u_1, \ldots, u_n$ and connects vertex u_1 with vertex u_n if there exist directed edges $(u_1, u_2), (u_2, u_3), \ldots (u_{n-1}, u_n)$.

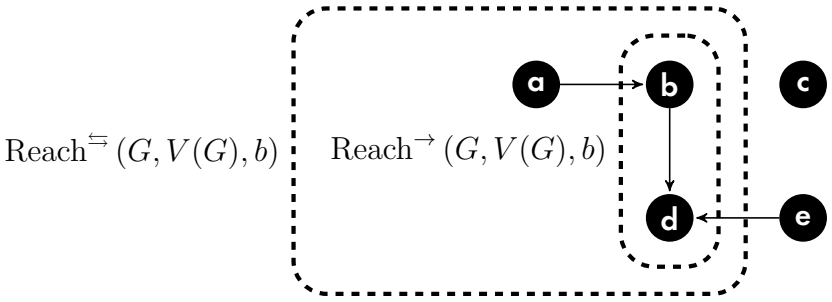
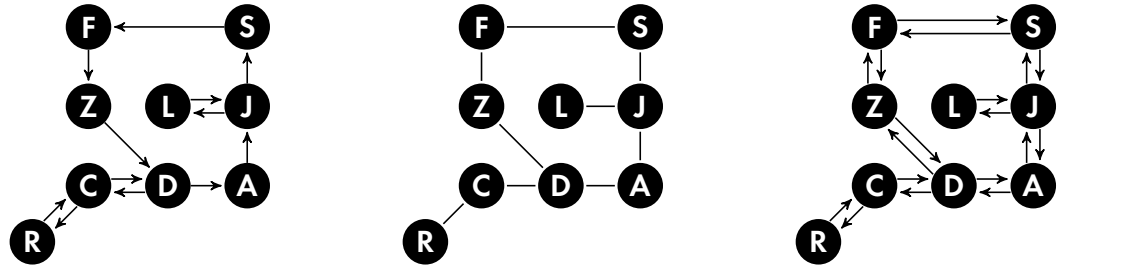


Figure 2.2.: A simple example of different reach measures on G

We will write $\mathrm{Reach}^{\rightarrow}(G, W, v) = \{u \in V(G) \mid \exists (v \rightsquigarrow u) \text{ in } G[W]\}$ for the set of vertices reachable by a path from v on the subgraph induced by W. Furthermore, we will use a similar definition for the set of vertices that are either reachable with a path from v or that are able to reach v , $\mathrm{Reach}^{\leftrightarrows}(G, W, v) = \{u \in V(G) \mid \exists (u \rightsquigarrow v \in G[W]) \vee \exists (u \rightsquigarrow v \in G[W])\}$. $\mathrm{Reach}(G, W, v)$ is the equivalent for undirected graphs, only ensuring a path between two vertices can be found. For the graph G in Figure 2.2, $\mathrm{Reach}^{\rightarrow}(G, V(G), b) = \{b, d\}$ while $\mathrm{Reach}^{\leftrightarrows}(G, V(G), b) = \{a, b, d\}$. If the graph was undirected, $\mathrm{Reach}(G, V(G), b)$ would be $\{a, b, d, e\}$.

A *complete graph* K_n contains all possible edges between its n vertices, that is, for any two distinct vertices $u, v \in V(G), u \neq v : (u, v) \in E(G)$. A one-way complete graph H_n may also contain just one edge $u, v \in V(H), u \neq v : (u, v) \in E(G) \vee (v, u) \in E(H)$. In general, $H_n \subseteq K_n$. For examples with $n = 8$, see Figure 4.1g on page 51. A k-*clique* is a completely subgraph of a graph with k vertices. A graph is *bipartite* if it consists of two sets that may be connected by edges between each other but none inside.

The underlying undirected graph of a graph G can be produced by creating an undirected graph U with $V(U) = V(G)$ and adding edges $E(U) = \{\{u, v\} \mid (u, v) \in E(G) \vee (v, u) \in E(G)\}$. Information is lost in this case. An undirected graph may be represented by a directed graph that has edges in both directions where the undirected edges would have been.



(a) Original boat course G (b) Undirected graph U (c) Converting U back to directed

Figure 2.3.: Conversion between directed and undirected graphs

Graph grammars (Kreowski et al., 2006) are a powerful tool for graph transformation.

> **Definition 1 (Graph grammar)** A *graph grammar* $GG = (\Sigma, S, P, T)$ for *labels* Σ consists of the *initial graph* $S \in \mathcal{G}_\Sigma$ (the set of all labeled graphs), *graph transformation rules* P and *allowed terminal labels* $T \subseteq \Sigma$. Graph transformation rules are a set of tuples (Γ, U, W) with $U, W \in \mathcal{G}_{\Sigma,\Gamma}$, the set of all possible graphs G annotated with labels Σ and additional per-transformation-rule labels Γ.
>
> A rule can be applied to a graph, replacing a subgraph U with a subgraph W. The empty graph is represented as ε. Per transformation rule, additional labels Γ may be used to re-identify vertices. When applying a graph grammar to a graph, we obtain a language $L(GG)$ of graphs that can be derived from S by applying the transformation rules and only consist of vertices labeled according to T.

As such, they are very similar to Chomsky grammars on strings. We will generally represent labels Σ by drawing vertices differently while per-rule labels Γ will be placed next to vertices.

For example, the following graph grammar could be used to describe all directed circles $L(GG)$:

$$\Sigma = \{v\}$$
$$H_3 = (\{a, b, c\}, \{(a, b), (b, c), (c, a)\})$$
$$GG = (\Sigma, H_3, \{(\{v_a, v_b, v_n\}, (\{v_a, v_b\}, \{(v_a, v_b)\})), (\{v_a, v_b, v_n\}, \{(v_a, v_n), (v_n, v_b)\})\}, \Sigma)$$

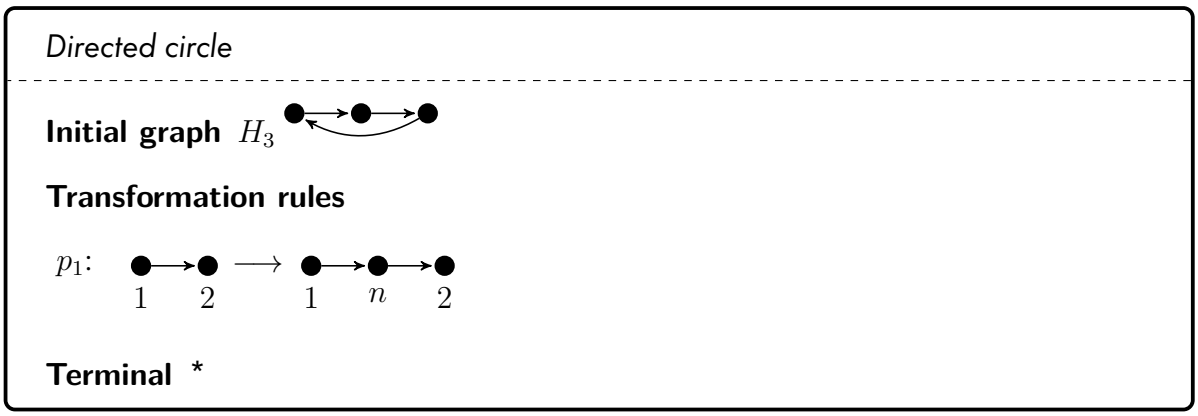
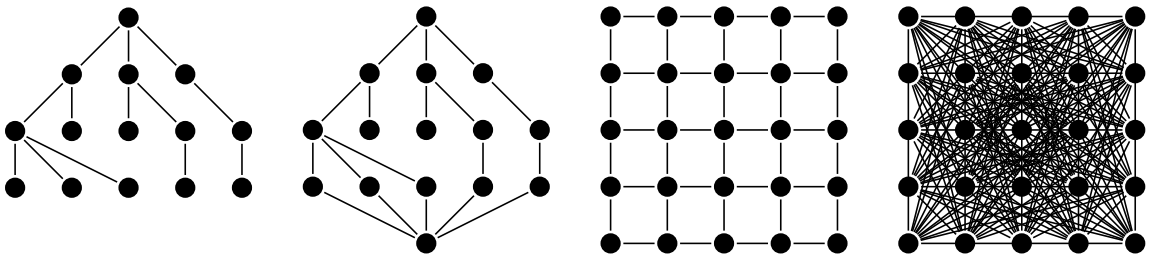


Figure 2.4.: Constructing directed circles

It can be drawn as in Figure 2.4. Notably, we do not need to apply labels to vertices and only use Γ in individual rules to prevent switching edge directions.

2.2. Tree-Width

Tree-Width for undirected graphs was discovered independently multiple times. In 1972, Bertelé and Brioschi introduced the concept of *dimension*. Halin (1976) later defined *S-Functions*, that work in the same way. The term Tree-Width was coined by Robertson and Seymour (1984) when they discovered it again and brought it to the attention of many mathematicians as it plays a key role in the celebrated graph minors project that eventually led to the proof of Wagner's conjecture (Robertson and Seymour, 2004). Initially, they defined Tree-Width by tree-decompositions, which we will define in Section 2.2.1 on the facing page. Since then, many other authors have studied Tree-Width as well, finding other equivalent ways to define it.



(a) A tree
Tree-Width is one

(b) Still similar to a tree
Tree-Width is two

(c) A grid
Tree-Width is five

(d) Complete graph
Tree-Width is 24

Figure 2.5.: Basic Tree-Width examples

It is defined on undirected graphs. Trees and forests always have a Tree-Width of one (Diestel, 2018). The less tree-like a graph is, the higher its Tree-Width rises. The complete graph has the highest Tree-Width, in effect as large as its number of vertices.

2.2.1. Tree-decompositions

As mentioned before, tree-decompositions were originally defined by Robertson and Seymour (1984). Tree-decompositions also lead directly to many algorithmic applications as one can, similar as on trees, design dynamic algorithms that work bottom up from the leaves to the root of the decomposition.

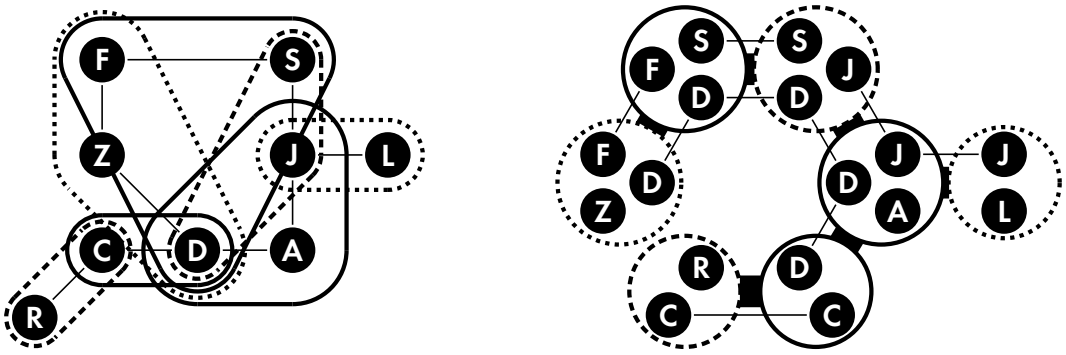
Definition 2 (Tree-decomposition) A *tree-decomposition* of an undirected graph G is a pair $(T, \mathcal{X})$. T is a tree and $\mathcal{X}$ is a family of subsets of $V(G)$ with $\mathcal{X} = \{X_t : t \in V(T)\}$ such that

- $\bigcup \mathcal{X} = V(G)$
- For every edge $e = (u, v) \in E(G)$ there exists a $t \in V(T) : u, v \in X_t$
- For $t, t', t'' \in V(T)$, with t' being on the path between t and t'': $X_t \cap X_{t''} \subseteq X_{t'}$

The *width* of such a tree-decomposition is $\max(\{|X_t| - 1 \mid t \in V(T)\})$.

G has a Tree-Width equal to the smallest width of all tree-decomposition that can be found.

Let us examine an example in Figure 2.6. A possible tree-decomposition consists of six decomposition sets (2.6a). Each of these sets allows us to cut the graph into pieces, just like removing a vertex from a tree. Furthermore, it is possible to draw the decomposition tree T itself (2.6b). The vertices $t \in V(T)$ are filled with the vertices that are in each respective set X_t. Dashed lines are used to help differentiate between sets. We can see that, for the Island of Ducks D which needs to be contained both in a set with the Zoo Z and the Arcade A, the path in the corresponding decomposition tree always contains D as well. As we are using up to three element sets, the Tree-Width of this graph is at most two.



(a) Decomposition sets X_t (b) Decomposition tree T

Figure 2.6.: Tree decompositions of the underlying undirected graph of Figure 2.3b on page 17

2.2.2. The invisible inert robber game for Tree-Width

In 1967, Breisch first introduced the concept of graph searching games, then motivated by his cave research.

> *The problem is this: A person is lost in a particular cave and is wandering aimlessly. Is there any efficient way for the rescue party to search for the lost person? What is the minimum number of searchers required to explore a cave so that it is impossible to miss finding the victim if [the lost person] is in the cave?* (Breisch, 1967, as cited in Dyer, 2018)

Graph searching games lead to one of the most intuitive and easy to understand alternative definitions of Tree-Width. There are several equivalent games, we are going to use the *invisible inert robber game*. Intuitively, a team of $k+1$ cops tries to capture a robber. The robber occupies some vertex of the graph. She is allowed to move if a cop is about to arrive and may then move to another vertex of the graph via a path that is not occupied by another cop. The cops on the other hand can move anywhere, then are not restricted by paths in the graph (imagine them to be in a helicopter).

The strategy the cops are going to use is known to the robber. In other words, she could observe all moves made by the cops and just announce having taken advantage of any movement pattern not accounted for by the cops, perhaps similar to a non-deterministic finite automaton (NFA) reaching an accepting state.

The Tree-Width becomes a measure of how many cops are needed to catch the robber, which very intuitively captures the notion of separators in a graph. As graphs become connected more strongly, the easier it gets for the robber to escape. This results in an increasing amount of cops required to be able to capture her – by taking advantage of the limited Tree-Width of the underlying graph.

Definition 3 (Invisible inert robber game for Tree-Width) Formally, the invisible and inert cops and robber game with $k+1$ cops can be described by series of n game states:

$$(C_1, R_1), (C_2, R_2), \ldots, (C_n, R_n)$$

Each tuple represents a set of up to $k+1$ cops locations $C_i \subseteq V(G)$ with $|C_i| \leq k+1$ and possible robber locations R_i for a specific move i according to the cops' strategy. Initially, the robber could be located everywhere: $(C_0, R_0) = (\emptyset, V(G))$

For every move, the $k+1$ cops are allowed to move freely: $C_i \subseteq V(G), |C_i| \leq k+1$.

Meanwhile, the robber is only allowed to move as soon as a cop is about to enter her location. She can move along paths in the graph as far as she likes, but she cannot use paths through vertices that are currently occupied by cops. The cop needs to leave their

position to start moving, she is able to use this freed vertex both for finding paths and resting. In the end, she will rest in a place that won't be occupied by a cop:

$$R_{i+1} = \left(R_i \cup \bigcup_{v \in R_i \cap C_{i+1}} \text{Reach}\left(G, V(G) \setminus (C_i \cap C_{i+1}), v\right) \right) \setminus C_{i+1}.$$

When looking at strategies for this game, the cops need to find a way to eliminate all possible robber locations. In fact, this is a one-player game. The robber does not have a strategy on her own, as the changes to the possible robber locations are completely determined by the new cop locations. The cop strategy is to find the right set of locations $C_1, \ldots, C_n$ to capture the robber. As a result, the cops have a winning strategy if they can reach a game state with $R_n = \emptyset$.

A strategy is robber-monotone if the robber is not able to occupy any vertex that was inaccessible before: $\forall i \in [2, \ldots, n-1] : R_i \subseteq R_{i-1}$. The restriction that the robber can only move upon arrival of a cop allows the strategy of cops choosing locations $C_1, \ldots, C_n$ to be simplified as a pair of choosing up to k locations B_i that cannot trigger a robber movement, and then occupying a new vertex c_i with the remaining cop. This results in an elimination order $c_1, \ldots, c_n$, that will become relevant later.

A lot of alternative cops and robber games have been defined. LaPaugh (1993) focused on contaminated edges. Seymour and Thomas (1993) have created an alternative game with a visible robber where fewer cops are sufficient as they don't need to account for parts of the graph that weren't chosen.

2.2.3. Partial k-trees

Tree decompositions illustrate how to deconstruct and analyze a graph. Partial k-trees on the other hand offer a constructive view on Tree-Width. Intuitively, they are easily described using graph grammars.

During construction, we always ensure that the Tree-Width doesn't exceed the limit we specified at the start. As long as we follow a set of rules, we can now transform any k-tree into a k-tree with more vertices.

Technically, when adding a new vertex, the positions we connect it with are the locations a cop would need to reside at for clearing that vertex. By remembering this order and because the robber may only move prior to cop occupation, these vertices remain exactly as the separators between the already cleared, and the still possibly occupied vertices of the k-tree. The basic principle of partial k-trees is that any subgraph will have a smaller or equal Tree-Width when compared to its super graph. We could, for instance, just use the very same cop strategy with the robber having a subset of its moves.

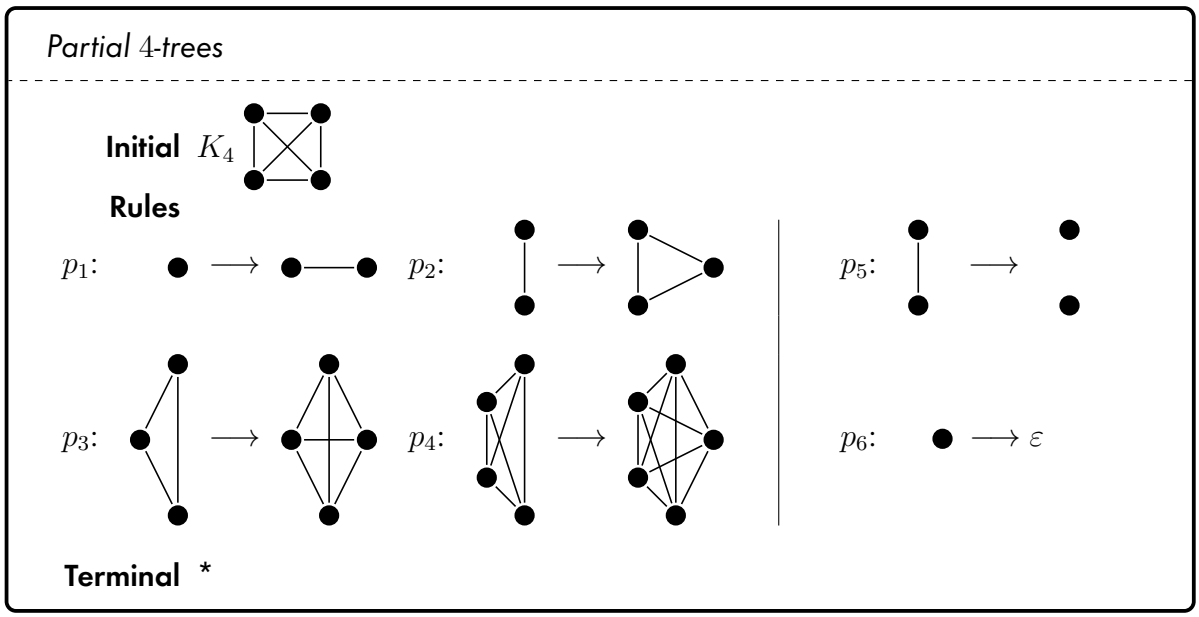


Figure 2.7.: Construction rules for partial 4-trees

Such a super graph may be created first by following a set of simple rules first devised by Rose in 1970:

> **Definition 4 (Partial k-tree)** For obtaining a k-tree for some k, first, the complete graph with k vertices K_k is created. It is a k-tree itself. After that, assume a k-tree G with n vertices is given, a k-tree G' with $n+1$ vertices may be created by adding a new vertex and connecting it with each member of a k-clique of G.
>
> Finally, vertices and edges of a k-tree may be removed to obtain a partial k-tree.

Using this procedure allows the creation of any graph with a Tree-Width of at most k. In Figure 2.7, we can see all rules that apply for $k = 4$. We have created additional rules for smaller inducing cliques, otherwise, we could have added support vertices to allow for this. We also allow removing edges right away as this does not increase the Tree-Width of the graph and does not remove restrictions for further construction steps.

2.2.4. Sparsity

Sparsity describes the amount of edges a graph has in relation to its vertices. Sparse graphs have fewer edges than dense graphs and are usually better suited for algorithms. One way to measure sparsity is the average degree of vertices of a graph (Nešetřil and Mendez, 2012).

> **Corollary 1** For graphs of fixed Tree-Width, the average degree of vertices is always at most twice as large as its Tree-Width.

Proof Strong induction over the number of vertices.

Base case At the beginning, each of the k vertices of the complete graph has an outgoing degree of $k-1$.

Induction hypothesis We assume that the average outgoing degree of k-trees with $n \geq k$ vertices is at most $2k$.

Induction step We will now determine the average degree of a k-tree with $n+1$ vertices. In each construction step, we add one vertex and up to k edges. As a result, the total degree rises by at most $2k$, as each edge is connected to two vertices. We now have an average degree of up to $\frac{2k \cdot n}{n} \cdot \frac{2k}{1} = 2k$ for a k-tree with $n+1$ vertices.

Those added edges are always connected to the added vertex. If remove one, we are not able to increase the degree over $2k$. □

As a result, a k-tree with n vertices can have at most $k \cdot n$ edges. A general graph on n vertices could have $\binom{n}{2}$ edges, which is quadratic in n.

2.2.5. Computing Tree-Width

Arnborg, Corneil, et al. (1987) demonstrated that given a graph, computing the smallest k such that a k-tree can be constructed for this graph is an NP-complete problem. There have been several efficient algorithms with different approaches, for example using parameterization or an approximation of a guaranteed quality.

2.3. Kelly-Width

There have been several attempts to generalize the definition of Tree-Width to directed graphs. The various characterizations lead to different notions for directed graphs. We will in particular study one measure related to the definition by orders, the invisible cops and robber game and k-trees at the same time.

Hunter and Kreutzer (2008) have discussed attempts of finding an equivalent measure for directed graphs. They have called it Kelly-Width after the famous outlaw Ned Kelly that was able to evade the police for several months, using a network of accomplices, safe hiding-places, and spies. In this section, we will present their aspects most relevant to understand the new Bi-Kelly-Width, which we will introduce in the next chapter.

Kelly-Width may as well be defined by a cops and robber game, explained in more detail in Section 2.3.1, which works similar to the game for undirected Tree-Width seen before. However, the robber is only allowed to move along edges in their direction.

For this, a directed graph is assumed with the new robber locations being computed via $\mathrm{Reach}^{\rightarrow}$ and with all other rules remaining unchanged.

Elimination orders are provided as a more formal approach directly usable as a winning strategy for the corresponding game. They are a rather direct adoption from the Tree-Width cops and robber game and transfer the rules to paths on directed graphs.

The new construction has properties similar to directed acyclic graphs (DAGs), so k-trees are adapted as k-DAGs. This includes their constructive approach and using partial k-DAGs to describe all graphs of width k. For this, the definition relies on slightly more conditions than the original:

k-DAGs always have a Kelly-Width of at most k. The initial graph is a full digraph with k vertices, each connected in both directions. Given a k-DAG with n vertices, a new k-DAG with $n+1$ vertices may be constructed by a set of simple construction rules.

We now define these concepts formally.

2.3.1. Cops and robber games

With the invisible inert robber game, Hunter and Kreutzer found a way to define Kelly-Width using a cops and robbers game. In this case, the robber is only allowed to move alongside edges in their natural direction, and only if the vertices are not occupied by cops.

Definition 5 (Invisible inert robber game for Kelly-Width) This game is played with $k+1$ cops and is played with a list of moves

$$(C_1, R_1), (C_2, R_2), \ldots, (C_n, R_n)$$

with sets of cops positions $C_i \subseteq V(G)$ with $|C_i| \leq k+1$ and possible robber positions $R_i \subseteq V(G)$ after a move i. We start with $(C_1, R_1) = (\emptyset, V(G))$. The cops win if $(C_n, R_n) = (C_n, \emptyset)$.

For a set of vertices C_i previously occupied, a number of cops may move to C_{i+1}. If any of these fields were occupied by the robber ($R_i \cap C_{i+1}$), she may start moving along edges as long as no remaining cops $C_i \cap C_{i+1}$ block them. The positions W a robber could move to starting at a specific vertex v are therefore limited to:

$$W = \mathrm{Reach}^{\rightarrow}\left(G, V(G) \setminus (C_i \cap C_{i+1}), v\right)$$

Therefore, a complete robber move from locations R_i to R_{i+1}, given previous cop positions C_i and future cop positions C_{i+1} may be defined as

$$R_{i+1} = (R_i \setminus C_{i+1}) \cup \left(\bigcup_{v \in R_i \cap C_{i+1}} \mathrm{Reach}^{\rightarrow}(G, V(G) \setminus (C_i \cap C_{i+1}), v) \right)$$

As the robber may only move if a cop would occupy her location, no changes will follow if no previous position is occupied:

$$(R_i \cap C_{i+1} = \emptyset) \Rightarrow (R_{i+1} = R_i)$$

Obviously, one cop can catch the robber on any acyclic digraph, hence, acyclic digraphs have Kelly-Width 0. In particular, also very dense graphs can have small Kelly-Width and Kelly-Width does not distinguish between complicated and simple acyclic graphs.

2.3.2. Elimination orders

Another formal approach to define Kelly-Width is via elimination orders. They naturally correspond to the monotone winning strategies in cops and robber games, however allow for a more intuitive approach to paths within graphs. This approach is slightly different to the original by Hunter and Kreutzer. We will present their idea in Section 2.3.3 on the following page instead.

Definition 6 (Strong reachability) Given a linear order $\leq = (v_1, v_2, \ldots v_n)$, we can define a new reachability measure on a graph. $\mathrm{SReach}^{\rightarrow}(G, \leq, v)$ denotes vertices that are smaller than v and that are reachable through a path with all of its internal vertices larger than v:

$$\mathrm{SReach}^{\rightarrow}(G, \leq, v) = \{w \in V(G) \mid w < v \text{ and } \exists p = v \rightsquigarrow w : \forall u \in p \setminus \{w\} : u \geq v\}$$

We decided to denote elements eliminated first as smaller. This may be defined in the opposite direction in some sources and should be noted to avoid confusion.

Definition 7 (Width of an order) The width of an order $\leq$ is the largest number of strong reachable vertices of any vertex.

$$\mathrm{Width}^{\rightarrow}(G, \leq) = \max\{|\mathrm{SReach}^{\rightarrow}(G, \leq, v)| \mid v \in V(G)\}$$

As such, they are closely connected to the inert robber game. Given an elimination order for a graph of width k, a winning strategy for k cops immediately follows. At position i, the cops have to move to the up to k smaller vertices in $\mathrm{SReach}^{\rightarrow}$. Afterwards, the one remaining cop

moves to v_i, removing it from the set of possible robber locations. Going by the definition of elimination orders, none of the previously occupied vertices can be reached anymore. They are always blocked by a cop. As a result, the robber can only reach the larger vertices. Therefore, the set of possible vertices is reduced for each inspected vertex. At the end of this game and following all moves, the graph is cleared of all possible robber locations, and the cops have won.

2.3.3. Fill-in graphs

Fill-in graphs were used by Hunter and Kreutzer (2008) as one of the three equivalent ways to define Kelly-Width. They used it as a formal approach to vertex elimination on digraphs. We should note, that our order is inverted to the one used by Hunter and Kreutzer. This allows us to be more consistent with cops and robber games.

Definition 8 (Fill-in graph) First, we start with a graph G and its vertices $V(G)$ placed in a linear order $\leq = (v_1, \ldots, v_n)$. Now, we construct the graph F with a fill-in procedure. It is called *fill-in graph* of G with respect to the order $\leq$.

Let G_n be G. For $i = n-1, n-2, \ldots, 1$, we will assume that the previous graph G_{i+1} has been constructed. Now, we obtain G_i from G_{i+1} by adding all edges (u, v) if $u, v < v_{i+1}$ and $(u, v_{i+1}), (v_{i+1}, v) \in E(G_{i+1})$. We apply these rules several times, until we reach G_1 at the end. This graph with all possible paths made explicit is now F.

The width of an order $\leq$ of $V(G)$ is equivalent to the maximum out-degree of any vertex v_i to smaller vertices in $G_i[\leq v_i]$, its respective fill-in step.

The vertices that the outgoing smaller edges of v_i in G_i lead to, directly correspond to $\mathrm{SReach}^{\rightarrow}(G, \leq, v_i)$. Hunter and Kreutzer called these vertices guards. Consequently, the Kelly-Width of a graph again corresponds to the order resulting in the smallest width – in this case a result of the fill-in procedure.

2.3.4. Partial k-DAGs

Similar to k-trees, k-DAGs can be defined using a small set of rules.

Definition 9 (Partial k-DAG) Again, we start with a complete directed graph of size k. Given an existing k-DAG H, we may add a new vertex v and then edges to v such that

- Up to k edges lead from v to a set $X \subseteq V(H)$
- We may then add edges (u, v) with $u \in V(H)$ if $(u, w) \in E(H)$ for all $w \in X \setminus \{u\}$

to obtain a new k-DAG of the same k. Now, a partial k-DAG is a subgraph of a k-DAG.

Any 0-DAGs and their subgraphs are DAGs. For $k' < k$, a k'-DAG is also a k-DAG, as we are allowed to apply the same rules.

Let us look at the construction of partial 4-DAGs in Figure 2.8. We always keep track of the currently added vertex to prevent adding illegal edges using a marker that is passed through creation steps.

2.3.5. Equivalence of concepts

All these measures are equivalent, as Hunter and Kreutzer (2008) demonstrated and proved:

Theorem 2 Let G be a digraph. The following are equivalent:

(1) G has a directed elimination ordering of width $\leq k$.

(2) $k + 1$ cops have a monotone winning strategy to capture an inert robber.

(3) G is a partial k-DAG.

2.3.6. Sparsity

Corollary 3 Graphs of bounded Kelly-Width are not necessarily sparse.

Proof As a counterexample, let us consider a specific class of one-way complete graphs. If we take a linear order of n vertices and connect any two distinct vertices by an edge leading from the smaller to the larger vertex, we end up with a graph free of directed circles and therefore a Kelly-Width of zero. This graph however, is still very dense with an average outgoing degree of $\frac{1}{2}n$. An example can be found in Figure 4.1f on page 51. □

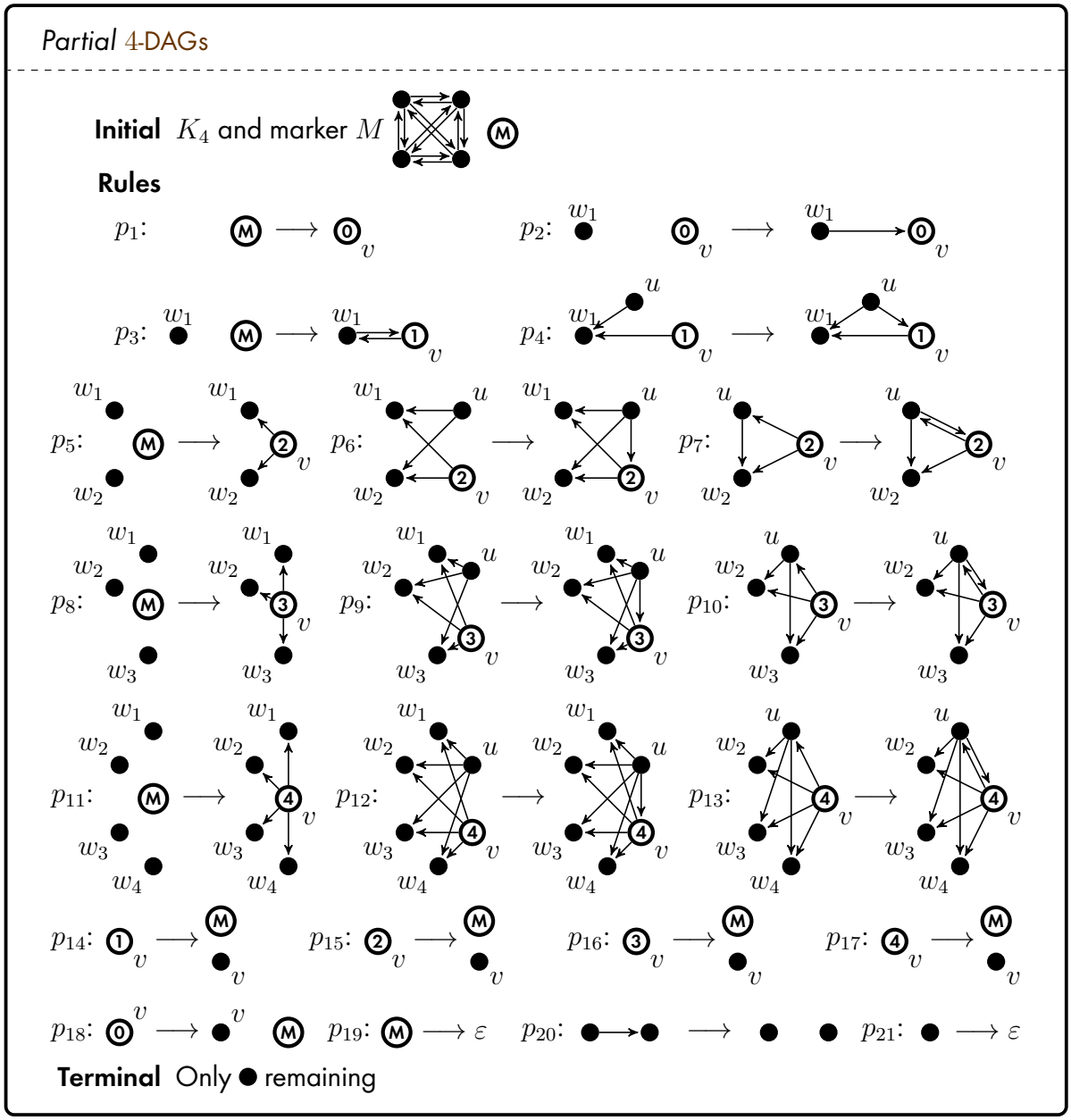


Figure 2.8.: Construction rules for partial 4-DAGs

3. Bi-Kelly-Width

In this chapter, we will introduce a new width measure for directed graphs, Bi-Kelly-Width. In contrast to Kelly-Width, it also takes the reverse direction of edges into account.

3.1. Cops and robber games

Cops and robber games for defining Bi-Kelly-Width can easily be extended from the inert invisible robber game used to characterize Kelly-Width in Section 2.3.1 on page 24. They are played on a directed graph. A limited number of $k + 1$ cops are allowed to move to any vertex. They are trying to capture an invisible robber who may only move once the cops are about to occupy its position. She, however, always knows the strategy of the cops and will always elude capture unless the cops follow a strategy that accounts for every starting position and every allowed move of the robber.

A further restriction is that the strategy of the cops needs to be monotone. They have to ensure that the robber is not able to reoccupy vertices where an occupation was ruled out earlier.

Once a cop announces its next position, the robber is allowed to start moving. In contrast to the cops that are allowed to move anywhere, she is only allowed to follow along edges, adhering to the direction of edges. However, at the start of a turn, she is also able to decide to follow edges in the opposite direction during her move. She is forced to only use one of both movement sets for each turn, but can use both over the course of the whole game. She may neither rest on nor move over vertices that are currently occupied by a cop. On the other hand, she is allowed to use locations that the cops have deserted, both in previous moves and, importantly, the vertices currently moving cops used to occupy.

As a result of our robber only being able to move once a cop arrives and having complete foreknowledge of the cop strategy, she rather becomes a set of possible locations. The cops always have to account for every possible location and for every movement that follows from landing on a possible location. Otherwise, she would have predicted this and decided to take this chance in the first place.

It is possible for the robber to reoccupy a vertex if a cop triggers a movement that has an unoccupied path to this vertex. As a result, if there are too few cops involved in the chase, she is able to find a set of densely connected vertices where she can always move between places the cops could occupy, each with enough connections to other hiding places. This is called a haven. In the worst case, the complete graph, each vertex would need to be occupied by a cop as the robber would always find somewhere to move to escape the moving cop if it were fewer. The whole graph becomes the haven.

3.1.1. Cops winning a game

Let us take a look at a complete game in Figure 3.1 on the next page. The game will be played by three cops on our boat course from Figure 1.2b on page 8. The graph consists of a directed circle with two branches to its sides that allow movement in both directions.

She is not able to escape, and the game is won by three cops in Figure 3.1o, though only two were involved in the last turns. Notably, even though the actual robber was located in the center right next to a previously cleared and not occupied vertex, she cannot move, as that would have required a cop announcing to moving there. This particular game did not involve reoccupation, but the next one in Section 3.1.2 will.

3.1.2. Robber winning a game

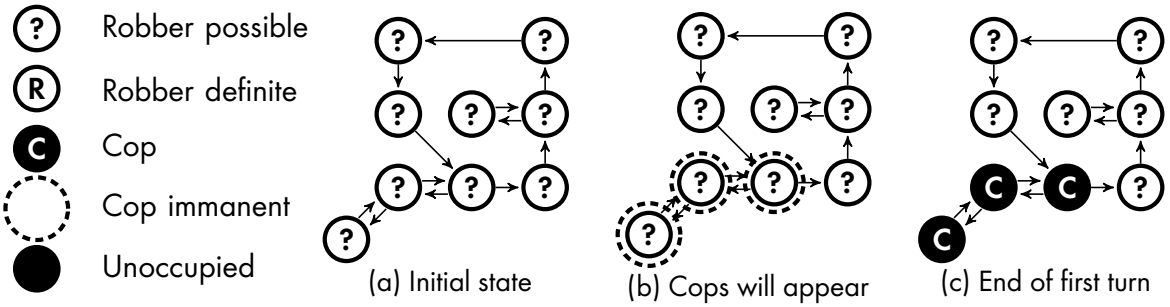
In Figure 3.2, we will try the same graph, using just two cops. Let us even assume that our robber has been so friendly to tell the cops where she will start and that she will only rest in three different places of her choice, while still following all other rules.

The cops can now move however they like, but they still face a problem. For any move they make, she is always able to escape, even though she keeps her promise, and the cops know exactly where she has to be.

It doesn't even make sense for the cops to move elsewhere. If they occupied one of the intermediate vertices on the path, she would even be able to reach both of the other appointed vertices once the other cop moved to her hiding place. Those three vertices form a haven for her, where she is able to evade the cops no matter how they move. In fact, she is always able to occupy a position previously owned by a cop if the cops move across the graph. Thus, she wins the game as the cops are never able to capture her.

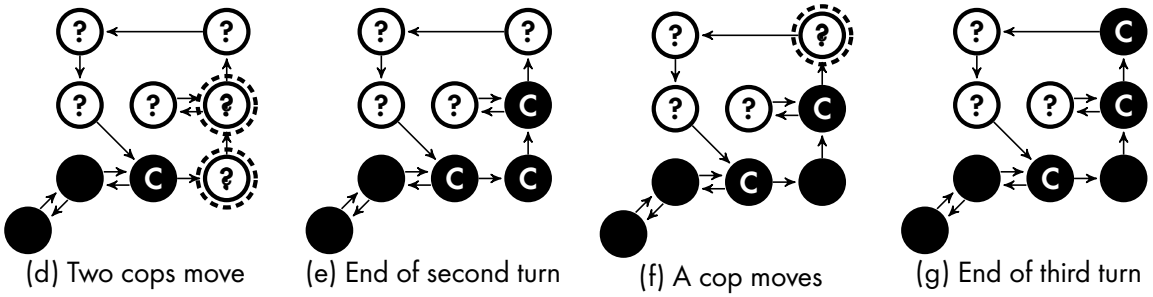
As a result, we now know that our boat course has a Bi-Kelly-Width of exactly two since we know that three cops have a monotone winning strategy while two can be evaded.

At the start of the game in 3.1a, the robber can be anywhere, so she has to be expected everywhere. 3.1b shows the first cops arriving. The robber could start moving from these positions, though there are no vertices she couldn't have occupied already, resulting in 3.1c.

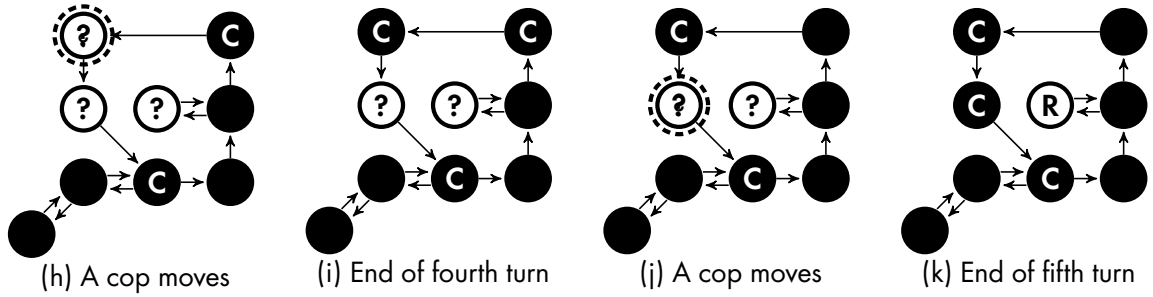


(a) Initial state (b) Cops will appear (c) End of first turn

For the next turns, one cop remains at the Ducks to guard the already cleared areas. Meanwhile, two cops take turns to clear the outer circle while also guarding the just cleared area.

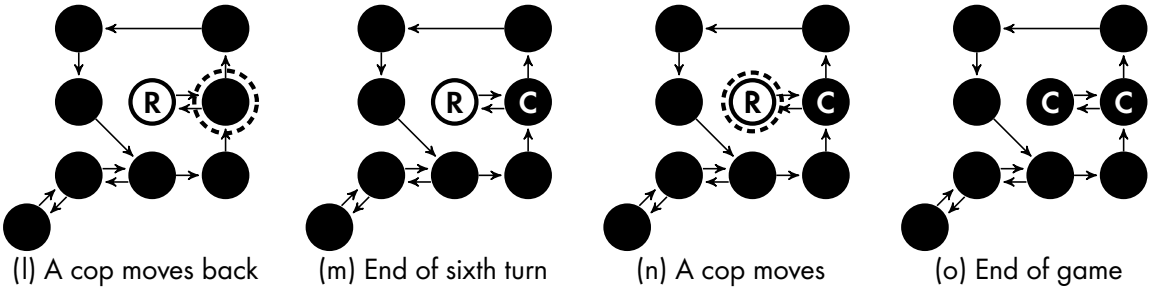


(d) Two cops move (e) End of second turn (f) A cop moves (g) End of third turn



(h) A cop moves (i) End of fourth turn (j) A cop moves (k) End of fifth turn

Finally, the last vertex she could have evaded capture on and therefore her hiding location needs to be inspected. The cops, however can't move there directly and need to guard the Junction first. In the end, as seen in 3.1n, a cop can announce the arrival at her location.



(l) A cop moves back (m) End of sixth turn (n) A cop moves (o) End of game

Figure 3.1.: Different turns as part of a game on the boat course from Figure 1.2b on page 8

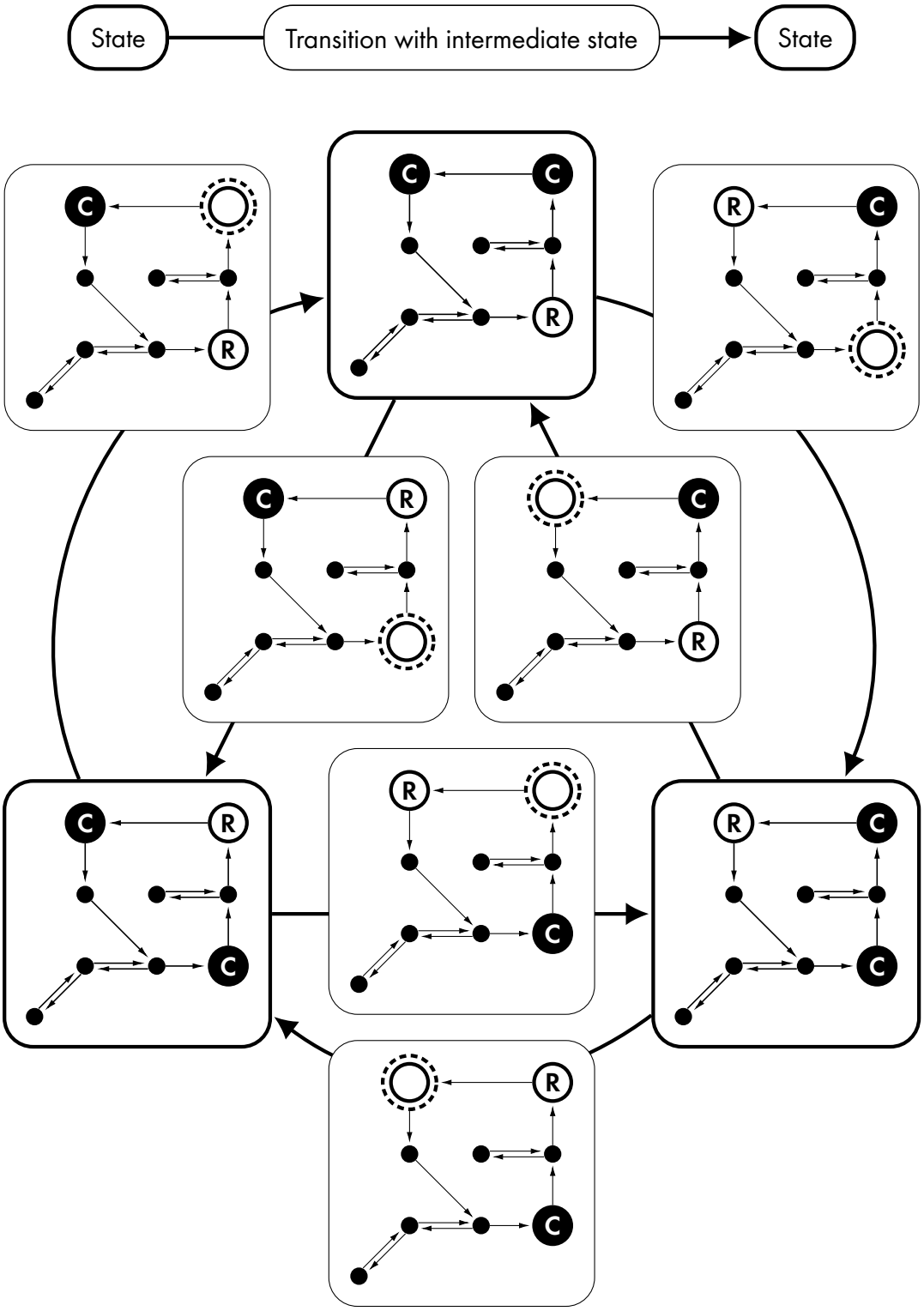


Figure 3.2.: States in an impossible game for two cops on the boat course from Figure 1.2b

Actually, any combination of three vertices of the circle have this property for only two cops. The haven exists if we can either find k vertices that are connected with disjoint paths to each other (so a cop cannot occupy a crucial vertex) or, in general, a location where all future moves and those of all successors will fulfill these conditions, even if the set of reachable vertices changes throughout the game. The latter occurs on complete grids as in Figure 4.4f.

3.1.3. Formal definition

Definition 10 (Invisible inert robber game for Bi-Kelly-Width) The invisible inert robber game is played on a graph G.

The game may be represented as a series of turns $(C_1, R_1), (C_2, R_2), \ldots, (C_n, R_n)$ consisting of cop locations $C_1, C_2, \ldots, C_n \subseteq V(G)$ and remaining possible robber locations $R_1, R_2, \ldots, R_n \subseteq V(G)$.

We again start with no cops and all vertices being the possible hiding place of our robber

$$(C_1, R_1) = (\emptyset, V(G))$$

and take no remaining places for our robber

$$(C_n, R_n) = (C_n, \emptyset)$$

as well as the monotonicity of the robber locations

$$\forall 1 \leq i \leq n : R_{i+1} \subseteq R_i$$

as the winning conditions for the cops.

For each move of the robber starting at a vertex v, she may move either along or against unoccupied edges, including those of cops moving in this turn:

$$\left\{w \in V(G) \setminus C_{i+1} \mid w \in \mathrm{Reach}^{\leftrightarrows}\left(G, V(G) \setminus (C_i \cap C_{i+1}), v\right)\right\}$$

If we combine this with the condition that the robber may only move if a cop would appear in her place, we are able to define all moves from possible locations R_i to the next allowed locations R_{i+1} for our robber:

$$R_{i+1} = \left(R_i \cup \bigcup_{v \in R_i \cap C_{i+1}} \mathrm{Reach}^{\leftrightarrows}\left(G, V(G) \setminus (C_i \cap C_{i+1}), v\right)\right) \setminus C_{i+1}$$

Now, we define the sufficient-guards-number k_{guards} for a graph such that we are able to find a monotone winning strategy for $k_{guards}+1$ cops. $|C_i| \leq k_{guards}+1$ for all $1 \leq i \leq n$.

3.1.4. Observations

Again, no changes in possible robber locations will follow if no possible robber location becomes occupied:

$$\begin{aligned} R_i \cap C_{i+1} = \emptyset \Rightarrow R_{i+1} &= \left(R_i \cup \bigcup_{v \in \emptyset} \mathrm{Reach}^{\leftrightarrows}(G, V(G) \setminus (C_i \cap C_{i+1}), v) \right) \setminus C_{i+1} \\ &= R_i \setminus C_{i+1} \\ &= R_i \end{aligned}$$

In general, graphs don't need to be connected. However, as a result of the movement patterns, the number of cops required for the component with the highest Bi-Kelly-Width would be able to clear this part of the graph and take care of all other components after that. This is because there is only one robber, and she is not able to move between components. It is notable that the component with the largest Bi-Kelly-Width is not necessarily the largest component. As a result, the Bi-Kelly-Width of a graph consisting of multiple components is equal to the maximum Bi-Kelly-Width of all components.

It is insignificant if a single or multiple cops are allowed to move at the same time. It is clear that even if multiple movements were allowed, the cops could also decide to always move alone, thus having the same result. On the other hand, as the moving cops disappear first and allow the robber to start moving from multiple locations, this would actually benefit the robber if used. In the end, neither the cops gain an advantage nor do they have to grant that for the robber unless they choose to.

3.2. Bi-directed elimination orders

In the following, we will discuss linear orders of vertices $\leq$ of a graph G. These lead to a width measure slightly different to $\mathrm{SReach}^{\rightarrow}(G, \leq, v)$ that was defined for Kelly-Width (Definition 6 on page 25).

Definition 11 (Strong bi-directed reachability) In this case, strong reachability takes into account paths in both directions. From a vertex v of a graph G given an order $\leq$, this includes all smaller vertices that can either be reached from v through a path through larger vertices or that can reach v only using vertices larger than v.

$$\begin{aligned} &\mathrm{SReach}^{\leftrightarrows}(G, \leq, v) \\ &= \{w \in V(G) \mid w < v \text{ and } \exists p \in \{v \rightsquigarrow w, w \rightsquigarrow v\} : \forall u \in p \setminus \{w\} : u \geq v\} \end{aligned}$$

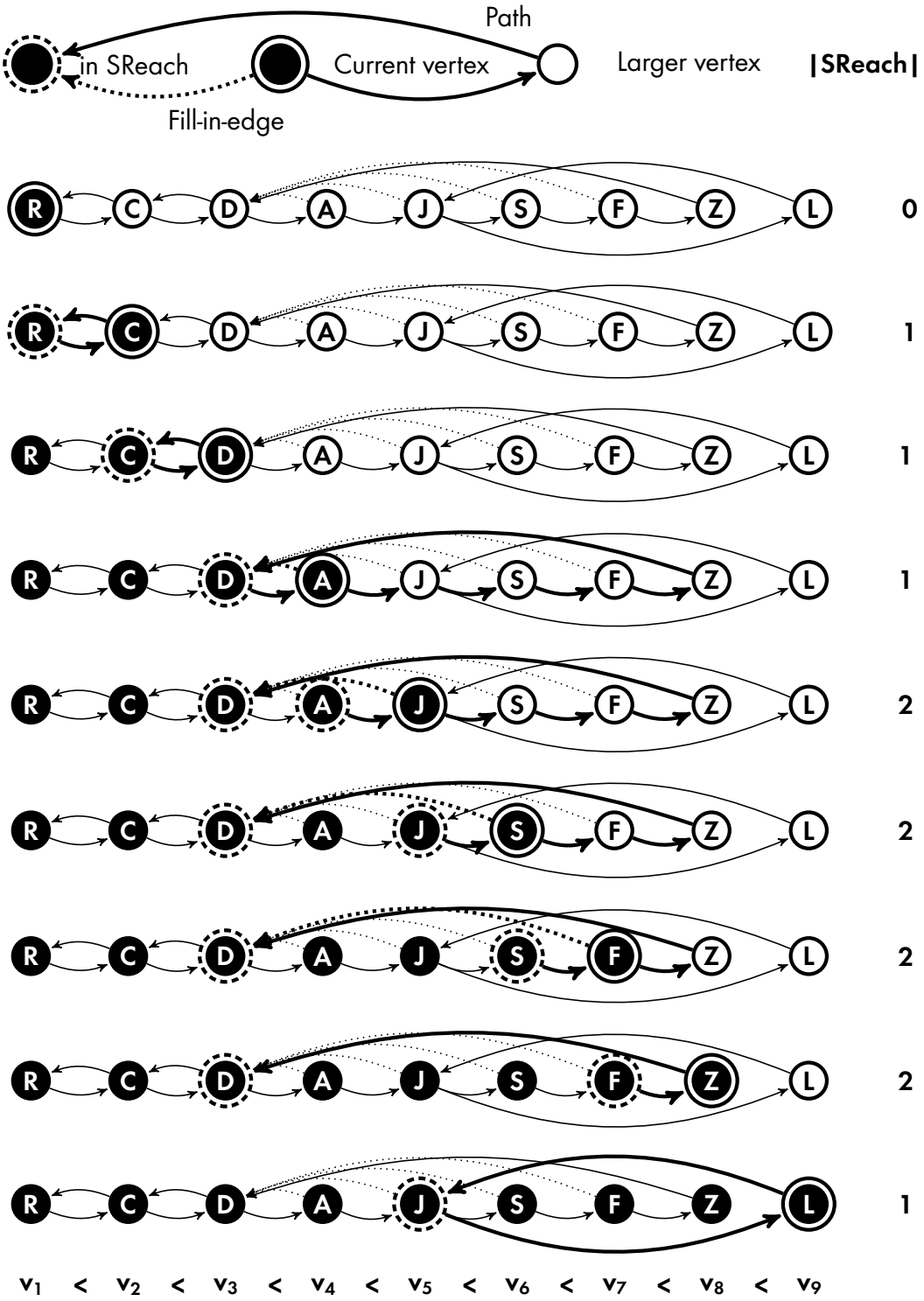


Figure 3.3.: Analyzing strong reachability for each vertex given an elimination order

Definition 12 (Bi-directed width of an order) The width of an order $\leq$ is the largest number of strong reachable vertices.

$$\mathrm{Width}^{\leftrightarrows}(G, \leq) = \max\left\{\left|\mathrm{SReach}^{\leftrightarrows}(G, \leq, v)\right| \mid v \in G\right\}$$

Definition 13 (Bi-Kelly-Width) The Bi-Kelly-Width of a graph is the smallest width of all possible orders.

$$\textsc{Bi-Kelly-Width}(G) \coloneqq \min\left\{\mathrm{Width}^{\leftrightarrows}(G, \leq) \mid \text{for all orders } \leq\right\}$$

If we look at our cops and robber game (Figure 3.1 on page 31), we will now treat early occupied vertices as smaller within our order, resulting in $(R, C, D, A, J, S, F, Z, L)$. In Figure 3.3 on the preceding page, $\mathrm{SReach}^{\rightarrow}(G, \leq, v_i)$ for each vertex is displayed. We can now take the maximum cardinality of these sets to calculate the width of this order, which is 2. For computing the actual Bi-Kelly-Width of the order, we would need try out all possible orders or use other measures that will be explained later. As there is no order resuting in a smaller width, 2 is also the Bi-Kelly-Width of this graph.

3.2.1. Fill-in graphs

Similarly to Kelly-Width, a fill-in graph with an equivalent Bi-Kelly-Width may be constructed. Only the restriction of added vertices is altered to encompass the additional possible paths to be taken care of.

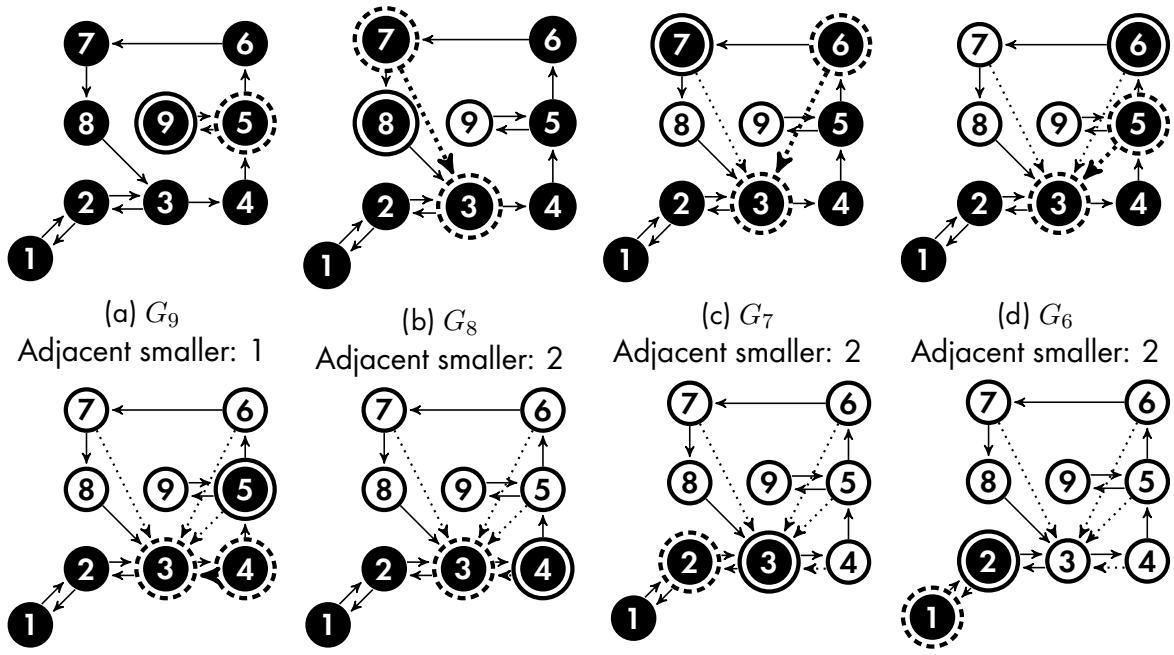
To do this, we will start with a graph G and its vertices $V(G)$ placed in a linear order of vertices $\leq$ or $(v_1, \ldots, v_n)$. Now, we are able to construct a new graph F with the fill-in procedure $F = \textsc{fill}(G, \leq)$. It is called *fill-in graph* of G with respect to the order $\leq$.

Definition 14 ($\textsc{fill}(G, \leq)$) Let G_n be G. For $i = n-1, n-2, \ldots, 1$, we will assume that the previous graph G_{i+1} has been constructed already. Now, we are able to obtain G_i from G_{i+1} by adding all edges (u, v) if $u, v < v_{i+1}$ and $(u, v_{i+1}), (v_{i+1}, v) \in E(G_{i+1})$. As we are able to apply these rules several times, we can reach G_1 at the end. This graph with all possible paths made explicit is now the fill-in graph $G_1 = F$.

Lemma 4 Let G be a graph and let $\leq$ be an order of $V(G)$. Then $\mathrm{SReach}^{\leftrightarrows}(G, \leq, v_i)$ exactly contains the set of smaller adjacent vertices of v_i in G_i (in either direction). As a result, the width of an order $\leq$ can be found in F as the highest number of smaller vertices adjacent to any vertex.

Proof Let us use complete induction over the inspected vertex i, counting down from n.

Base case In the base case, $i = n$. Therefore, there are no larger vertices than v_i. As a result, $\mathrm{SReach}^{\leftrightarrows}(G, \leq, v_i) = \{v \mid (v, v_i) \in E(G) \vee (v_i, v) \in E(G), v \neq v_i\}$. At the same time, no vertices have been added to the fill-in graph yet, so both sets are equal.



(a) G_9
Adjacent smaller: 1

(b) G_8
Adjacent smaller: 2

(c) G_7
Adjacent smaller: 2

(d) G_6
Adjacent smaller: 2

(e) G_5
Adjacent smaller: 2

(f) G_4
Adjacent smaller: 1

(g) G_3
Adjacent smaller: 1

(h) G_2 (same as G_1)
Adjacent smaller: 1

Figure 3.4.: Fill-in graph construction for the boat course

Induction hypothesis We now assume that Lemma 4 applies to all vertices $\geq v_{i+1}$ and G_{i+1}.

Induction step We will now construct G_i and inspect the vertex v_i.

> For each vertex $v > v_i$, $\mathrm{SReach}^{\leftrightarrows}(G, \leq, v)$ now contains all vertices that have the complete internal path from/to v in $G[> v]$. For G_{i+1}, all these have been connected by an edge. G_i now adds a shortcut for vertices that were connected through a directed path in G by adding vertices according to connections in G_{i+1}. The smaller vertices connected with v_i in G_i are therefore exactly $\mathrm{SReach}^{\leftrightarrows}(G, \leq, v_i)$. Furthermore, this step does not remove connections of vertices in $G[< v_i]$.

Therefore, Lemma 4 on the preceding page applies to all steps leading up to G_1. Moreover, $\mathrm{SReach}^{\leftrightarrows}(G, \leq, v_i)$ is exactly $\mathrm{SReach}^{\leftrightarrows}(F, \leq, v_i)$ for all $v_i \in V(G)$. □

> **Corollary 5** As a result, the width of an order may also be determined with a fill-in graph, resulting in an alternative characterisation for Bi-Kelly-Width. We will look at the adjacent vertices in the fill-in graph.

$$\mathrm{Adj}^{\leftrightarrows}(G, v) = \{v \in V(G) \mid \exists\,(v, w) \in E(G) \vee \exists\,(w, v) \in E(G)\}$$

We take the largest amount of smaller adjacent vertices in both directions as the width of our order.

$$\begin{aligned}&\text{Width}^{\leftrightarrows}(G, \leq) = \max\{\\ &\quad |\{w \in V(G) \mid w \in \text{Adj}^{\leftrightarrows}(\textsc{fill}(G, \leq), v), w < v\}|\\ &\mid \text{for all } v \in V(G)\}\end{aligned}$$

Now we need to find the order with the smallest width.

$$\textsc{Bi-Kelly-Width}(G) = \min\left\{\text{Width}^{\leftrightarrows}(G, \leq) \mid \text{for all orders} \leq\right\}$$

v_i may also be taken from G_i as edges to smaller vertices aren't added afterwards anymore.

In Figure 3.4 on the preceding page, we create the fill-in graph using the same order as in the previous section, $(R, C, D, A, J, S, F, Z, L)$. The additional edges are also displayed in Figure 3.3, where the equivalence of both notions becomes even more apparent. Again, the width of the boat course using this order is 2, the maximum of smaller connected vertices.

3.3. Partial bi-directed k-trees

Similar to Kelly-Width with its partial k-DAGs, the partial bi-directed k-tree can be constructed with a few simple rules.

First, we are able to construct a bi-directed k-tree:

Definition 15 (Partial bi-directed k-tree) A complete directed graph of k vertices, or in other words a graph with k vertices that are connected to each other, is a bi-directed k-tree.

A bi-directed k-tree with $n+1$ vertices can be constructed by taking a bi-directed k-tree H with n vertices and adding a vertex v. In the next step, we will keep in mind two subsets, $X^{\rightarrow}, X^{\leftarrow} \subseteq V(H)$ with $|X^{\rightarrow} \cup X^{\leftarrow}| \leq k$ such that any two vertices $u \in X^{\rightarrow}, w \in X^{\leftarrow}$ are either connected by an edge (w, u) or (u, w) or both. We will now add new edges

- (u, v) for all $u \in X^{\rightarrow}$
- (v, w) for all $w \in X^{\leftarrow}$

A partial bi-directed k-tree simply is a subgraph of a bi-directed k-tree. As the presence of edges in a step doesn't impose new restrictions itself - we are only allowed to add more edges if edges are present already - a partial bi-directed k-tree may also be constructed by only adding a subset of edges in each step, as long as all previous conditions still hold.

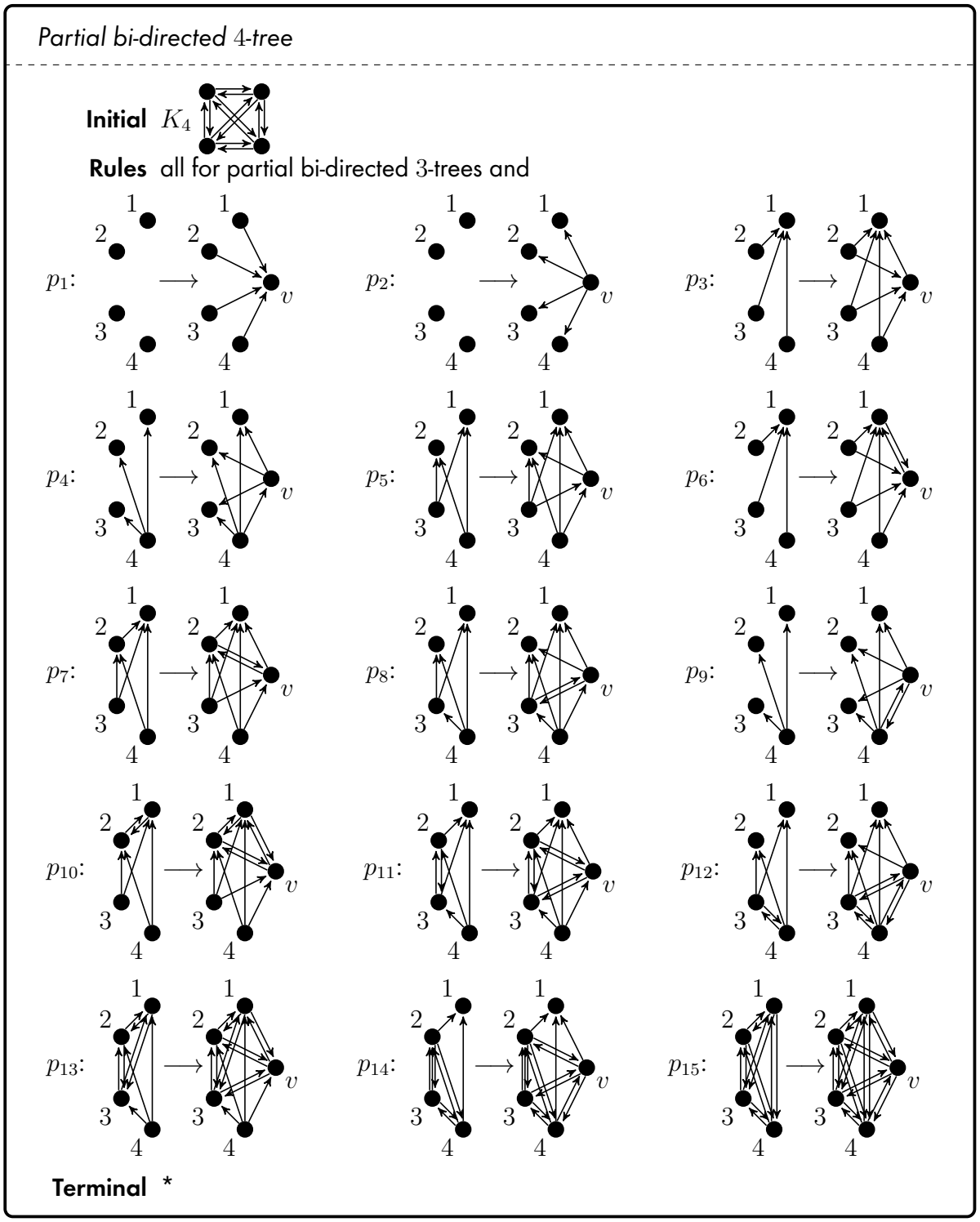


Figure 3.5.: Construction rules for partial bi-directed 4-trees

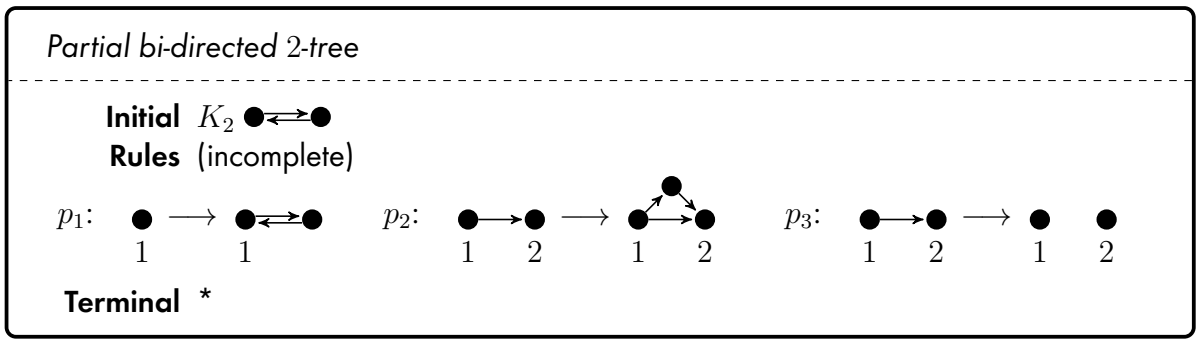


Figure 3.6.: Construction rules sufficient for boat course construction

3.3.1. Observations

The two subsets $X^{\rightarrow}, X^{\leftarrow} \subseteq V(H)$ actually need to be both sides of an at least one-way complete bipartite graph during each step. This doesn't rule out other connections between members within a set. If there is an intersection $X^{\rightarrow} \cap X^{\leftarrow}$ between the two subsets, this subset must be a two-way clique.

They don't necessarily need to have the same size, as long as their total size doesn't exceed the desired Bi-Kelly-Width.

Moreover, if $X^{\rightarrow} = X^{\leftarrow}$, we obtain the same condition of extending a k-clique as with Tree-Width in Section 2.2.3. This again demonstrates how close this measure is to the original notion of Tree-Width.

3.3.2. Constructing a partial bi-directed k-tree

We will now recreate our boat course in Figure 3.7. As we already know that it will have a Bi-Kelly-Width of two, we will use this as our limit for subsets $|X^{\rightarrow} \cup X^{\leftarrow}| \leq 2$. Because of the relatively simple structure of the boat course, we actually only needed a subset of rules for partial bi-directed 2-trees, Figure 3.6.

We start with a complete graph with two edges in Figure 3.7a. In Figures 3.7b, 3.7c and 3.7h, we chose $X^{\rightarrow} = X^{\leftarrow}$ with a size of one. Figures 3.7d to 3.7g all use the previous vertex as the one-element $X^{\rightarrow}$ and $X^{\leftarrow}$ to contain the Island of Ducks. This still ensures the cardinality of their union to be 2. With the constructed bi-directed k-tree Figure 3.7i, we can remove some unnecessary edges (Figure 3.7j) to obtain our desired graph Figure 3.7k.

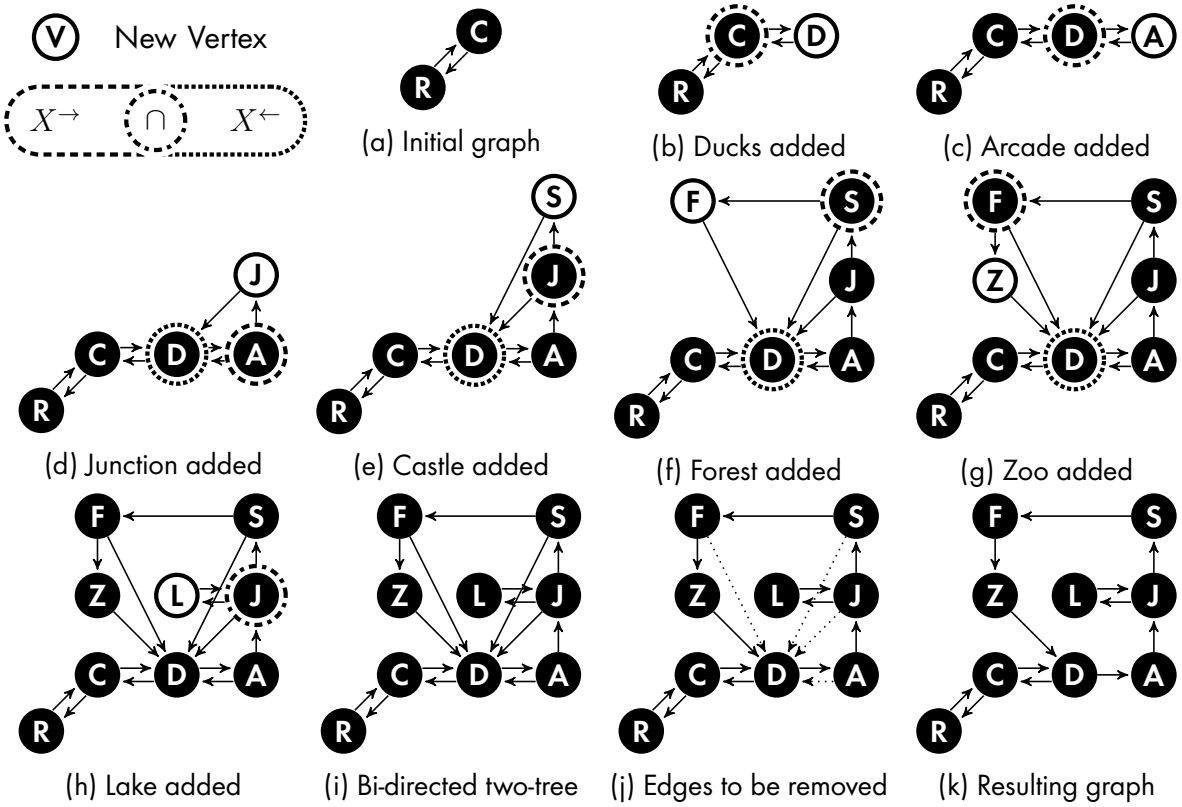


Figure 3.7.: Constructing the boat course in the Bürgerpark

3.3.3. Sparsity

Corollary 6 As with Tree-Width (Section 2.2.4 on page 22), graphs of bounded Bi-Kelly-Width are sparse. Their average outgoing degree is at most $2k$.

Proof The average outgoing and incoming degree of a graph are the same, so we will only look at the outgoing degrees. We will use complete induction over the number of vertices.

Base case At the beginning, each of the k vertices of the complete graph may have an outgoing degree of $k-1$ each.

Induction hypothesis We assume that the average outgoing degree of bi-directed k-trees with $n \geq k$ vertices is at most $2k$.

Induction step We will now determine the average degree of a bi-directed k-tree with $n+1$ vertices. During construction, we may choose sets $X^{\rightarrow}$, $X^{\leftarrow}$ with a total of k vertices. As a result, up to $2k$ edges are added during this construction step. We now have an average degree of up to $\frac{2k \cdot n}{n} \cdot \frac{2k}{1} = 2k$ for a bi-directed k-tree with $n+1$ vertices.

The added edges rely on each individual vertex, so a partial k-tree cannot increase its average degree over $2k$ by removing specific vertices. □

3.4. Equivalence of concepts

As with Kelly-Width, all three concepts are equivalent. The elimination order with its fill-in graph can be interpreted as the steps required in a cops and robber game (Section 3.1). All elimination orders (Section 3.2) are constructed procedurally. The game can be played and won on the graphs constructed procedurally (Section 3.3).

Theorem 7 Given a graph G and a fixed k the following are equal:

- k is a sufficient-guards-number in the cops and robber game from Definition 10.
- An elimination order with a width k can be found both in terms of $\mathrm{SReach}^{\leftrightarrows}$ and for fill-in graphs.
- G is a partial bi-directed k-tree.

Hence, the k used in orderings used to analyze strong reachability or to create fill-in graphs, partial bi-directed k-trees and the sufficient-guards-number k_{guards} for $k+1$ cops being able to catch a robber are all an upper bound to the Bi-Kelly-Width of a graph. The smallest k that can be found in any of these representations is the Bi-Kelly-Width of that graph.

Proof The proof can be performed similarly to Kelly-Width (Hunter and Kreutzer, 2008), only taking into account the differences introduced before. The equivalence of definitions for elimination orders was already shown in Lemma 4.

3.4.1. Cops and robber games → Elimination orders

Let us assume, that $k+1$ cops are sufficient to find a robber monotone winning strategy. As it makes no difference, see Section 3.1.4, we will assume that this winning strategy relies on moving each cop individually.

Let us now order all vertices by their first occupation by a cop in our winning strategy. We will call this order $\leq$ or $(v_1, v_2, \ldots, v_n)$. v_1 is the first vertex to be visited while v_n will be the last that will conclude the game.

We will now claim that this order has a width of at most k. Let us look at the movements in the game. As our $k+1$ cops have a monotone winning-strategy, no occupation of a new vertex allows the robber to move to a previously already occupied and cleared vertex. To help with this, we have up to k other cops that can be moved within the set of already cleared vertices. As

a result, these will always be placed in a way that obstructs hypothetical paths for the robber to escape back into the cleared part of the graph. If she in fact could escape, the strategy couldn't have been monotone in the first place, contradicting the original assumption.

These vertices however now form $\mathrm{SReach}^{\leftrightarrows}(G, \leq, v_i)$. They are smaller with respect to the order, and are the up to k vertices that are placed on paths that need to be blocked as they directly come from or lead to the newly occupied vertex.

3.4.2. Elimination orders $\rightarrow$ Partial bi-directed k-trees

Let us consider the fill-in graph F for a linear order $\leq = (v_1, v_2, \ldots, v_n)$ of graph G that leads to the width k. We will claim that the subgraph induced by the last $k + j$ vertices of this fill-in graph are always a subgraph of a bi-directed k-tree T_j.

Let us prove this via complete induction over j.

Base case For $j = 0$, let us consider the last k vertices. The induced subgraph of those k vertices is always a subgraph of a complete graph on k vertices.

Induction hypothesis Let us assume that the last $k+n$ vertices of the fill-in graph are a subgraph of a bi-directed k-tree T_n.

Induction step For $j = n+1$, we will inspect the vertex $v_{|V(G)|-(k+n+1)}$ that needs to be added for T_{n+1}. In other words, $i = |V(G)| - (k + n)$.

As a result of the construction of $\textsc{fill}(G, \leq)$ only being connected to k lower vertices, these form our at most k vertices the new vertex $v_{|V(G)|-(k+n+1)}$ is being connected to. The second property, that for fixed sets $X^{\rightarrow}, X^{\leftarrow}$,

- (u, v) for all $u \in X^{\rightarrow}$
- (v, w) for all $w \in X^{\leftarrow}$

is automatically fulfilled as a result of our fill-in graph construction where we added all edges (u, v) if $u, v < v_{i+1}$ and $(u, v_{i+1}) \in E(G_{i+1})$ and $(v_{i+1}, v) \in E(G_{i+1})$.

Each edge either

- existed. No further steps were required.
- was added during construction. This however always ensures that a vertex between our two sets $X^{\rightarrow}, X^{\leftarrow}$ has been formed.

Therefore, our fill-in-graph is always a subgraph of a bi-directed k-tree, by definition a partial bi-directed k-tree.

3.4.3. Partial bi-directed k-trees $\rightarrow$ Cops and robber games

First, let us assume, that our bi-directed k-tree is not partial. We will now define a monotone winning strategy for each construction step using complete induction over the number of vertices n to be inspected.

Base case For a bi-directed k-tree with k vertices, we will place k cops on all initial vertices. This will definitely remove any possible robber.

Induction hypothesis Let us assume that the cops have a monotone winning strategy on the bi-directed k-tree with n vertices.

Induction step If we want to construct a bi-directed k-tree with $n+1$ vertices and add a vertex v. We will assume that all vertices that were added earlier are still occupied. We will now place a cop on any vertex in $X^{\rightarrow}$ and $X^{\leftarrow}$. This will require at most k cops as, per construction, $|X^{\rightarrow} \cup X^{\leftarrow}| \leq k$. As per induction hypothesis, this will not trigger any moves by our robber as these vertices were previously unoccupied. For the next move, we will place the last cop on v.

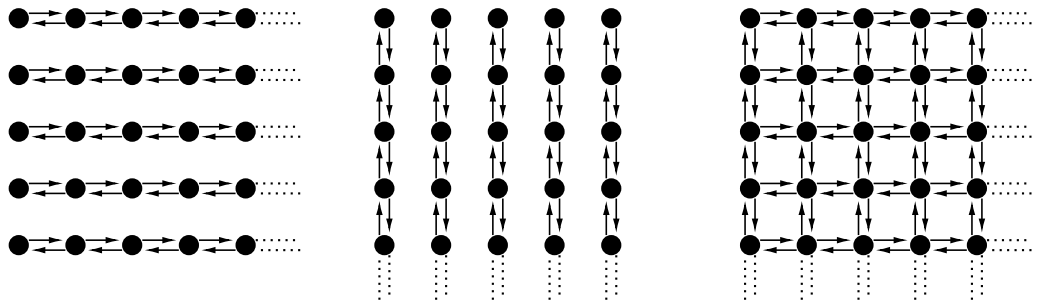
It is obvious that previous vertices cannot be reoccupied using the edges added during this step, but what about possible future edges and vertices? This is where the next condition, the one-way complete bipartite graph between all vertices in $X^{\rightarrow}$ and $X^{\leftarrow}$ comes in. During construction of any future edges for a vertex v', and therefore any future paths, we would have needed an edge between the future $X'^{\rightarrow}$ and $X'^{\leftarrow}$. If either of these vertices would need to be v, any future vertices v' would be connected to current members of $X^{\rightarrow}$ and $X^{\leftarrow}$. If they weren't, they must have been part of the same set where a path to follow won't be created. The robber would therefore have to stop at v' before being able to move again where blocking these exact moves can be accounted for.

As a result, we can find a monotone winning-strategy on any bi-directed k-tree such that $k+1$ cops are sufficient.

If we only take a partial bi-directed k-tree, the robber would only be allowed to use a subset of moves. The winning strategy of cops would not be affected negatively.

3.4.4. Conclusion

We have demonstrated that all individual properties bind the next observed to be equal or smaller that the initial size k. If we conclude the circle, we gain that $k_{elimination} \leq k_{k-trees} \leq k_{guards} \leq k_{elimination}$ and that all of them are, in fact, equal.



(a) Horizontal lines, constant Bi-Kelly-Width (b) Vertical lines, constant Bi-Kelly-Width (c) Combined graph with unbounded Bi-Kelly-Width

Figure 3.8.: An example for creating a graph with unbounded Bi-Kelly-Width from a union

The smallest k was defined as Bi-Kelly-Width for elimination orders (Section 3.2 on page 34) and therefore, Bi-Kelly-Width may be characterized by any of these measures. □

3.5. Closure properties

A lot of typical closure properties cannot be applied to Bi-Kelly-Width. For example, if we take two graphs with k lines of vertices (both having a Bi-Kelly-Width of one) and look at the union of both, we would obtain a grid of unbounded Bi-Kelly-Width k.

However, some interesting closure properties remain to be observed.

3.5.1. Subgraphs

Corollary 8 A subgraph $H \subseteq G$ has a Bi-Kelly-Width smaller than or equal to G.

$$\text{Bi-Kelly-Width}\,(H) \leq \text{Bi-Kelly-Width}\,(G)$$

The winning strategy of the associated cops and robber game would remain intact – the robber is restricted to less possible moves while the same number of cops can occupy the (remaining) vertices as before without opening additional paths for escaping.

3.5.2. Intersection

Corollary 9 For the intersection of two graphs $G \cap H$, the intersection of the graphs has a width at most as large as the smaller graph.

$$\begin{aligned}&\textsc{Bi-Kelly-Width}\,(G \cap H)\\ &\qquad \leq \min\left(\textsc{Bi-Kelly-Width}\,(G)\,, \textsc{Bi-Kelly-Width}\,(H)\right)\end{aligned}$$

This is a direct result of Corollary 8 on the previous page because the intersection is a subgraph of both graphs.

3.5.3. Limited union

In Section 3.1.4 on page 34, we have observed that for graphs with subcomponents that are not connected, the Bi-Kelly-Width only takes the worst of both into account.

Corollary 10 For a union of two graphs G, H with at most $h = |V(G) \cap V(H)|$ shared vertices, the width will be at most as large as h added to the Bi-Kelly-Width of the graph with the greater Bi-Kelly-Width.

$$\begin{aligned}&\textsc{Bi-Kelly-Width}\,(G \cup H)\\ &\qquad \leq\ \max\left(\textsc{Bi-Kelly-Width}\,(G)\,, \textsc{Bi-Kelly-Width}\,(H)\right) + |V(G) \cap V(H)|\end{aligned}$$

In the relevant cops and robber game, we would simply place additional cops on the h vertices and then continue with the existing strategies for each individual graph using the amount of cops required for that.

The same applies to adding h edges between two existing disjoint graphs or h new vertices that are the point of connecting any number of disjoint graphs.

3.5.4. Topological minors

A topological minor H in terms of Bi-Kelly-Width of a graph G is a graph with a subset of vertices $V(H) \subseteq V(G)$ and a set of edges such that all edges between vertices in H correspond to paths in G. These paths need to be disjoint, and may not share edges. The Bi-Kelly-Width of G will always be greater than or equal to the Bi-Kelly-Width of H.

This is a result of the cops and robber game definition. The same amount of cops can find a winning strategy on H by eliminating all vertices in the same order. In cases where paths to already cleared vertices exist, these need to be blocked by cops. As paths need to be disjoint,

and individual edges would need to have existed earlier, this can still only apply to k vertices. Otherwise, the robber could have escaped the winning strategy of the cops on our original graph G.

This allows finding a lower bound for Bi-Kelly-Width on graphs. We now just need to find a respective minor of a known size. Meanwhile, the upper bound can be provided by specifying an elimination order, a successful cops and robber game or a construction via partial bi-directed k-trees.

This construction differs from the Directed Topological Minors introduced in Ganian, Meister, et al. (2016) for DAG-Width and Kelly-Width. They do not take paths into account and, for example, allow the creation of completely new paths.

3.6. Comparison to other width measures

Theorem 11 The Bi-Kelly-Width of a graph G is always between its Kelly-Width and the Tree-Width of its underlying undirected graph U.

$$\textsc{Kelly-Width}(G) \leq \textsc{Bi-Kelly-Width}(G) \leq \textsc{Tree-Width}(U)$$

In general, Kelly-Width, Bi-Kelly-Width and the underlying Tree-Width of a graph are not identical.

Proof This is a result of the definition of the three cops and robber games.

Between Kelly-Width and Bi-Kelly-Width, the movement pattern of $k+1$ cops in the game for Bi-Kelly-Width can always be equally applied in the game for Kelly-Width because the robber is only allowed a subset of moves while all other factors remain identical. The additional moves allowed for the robber in games for Bi-Kelly-Width can require more cops.

When we look at the Bi-Kelly-Width-game for a symmetric digraph, our robber is able to use any non occupied path, just as the Tree-Width-robber on the underlying undirected graph. On the other hand, any just one-way directed edge restricts the robber movement without hindering cops, resulting in a possibly smaller Bi-Kelly-Width.

Figure 3.9 provides an example for non-equivalence. For an ordered 5×5-grid, Kelly-Width is 0, Bi-Kelly-Width is 2 and the underlying Tree-Width is 5. □

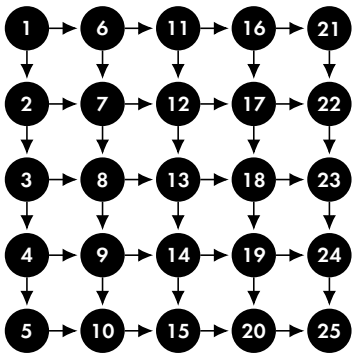


Figure 3.9.: An example for a difference between width measures

3.7. Computing Bi-Kelly-Width

Corollary 12 Computing the Bi-Kelly-Width of a graph is an NP-complete problem.

Proof First, we have to demonstrate that finding the Bi-Kelly-Width is in NP. We can easily construct a token that can be verified in polynomial time. It consists of an elimination ordering (verifiable in $O(n^3)$) and a topological minor of known same size and how to construct it. This can be verified by comparing paths in both graphs and is possible within $O(n^2)$ if we were to mark used edges in an adjacency matrix. Lastly, the width of the minor could have been demonstrated by a haven with k to $\min(k^2, n)$ different states, each with up to n adjacent states to verify the transition to.

We can now demonstrate that finding the Bi-Kelly-Width of a graph is at least as hard as other problems in NP. As computing the minimum k such that a k-tree can be constructed for a graph is NP-complete (Section 2.2.5), the same applies for finding the smallest possible width k of an order for Bi-Kelly-Width. We can simply create a polynomial time reduction on the Partial k-Tree Problem:

- For an undirected graph U, create a directed graph G with $V(G) = V(U)$ and $\{u, v\} \in E(U) \Rightarrow (u, v), (v, u) \in E(G)$.
- Find an elimination order with $\mathrm{Width}^{\leftrightarrows}(G, \leq) = k$, so $k + 1$ cops can monotonously catch the robber.
- The game can be played and won by an identical amount of cops in the cops and robber game for Tree-Width, as the allowed movement patterns are identical.
- The Tree-Width of this graph is at most k, as defined by the game.
- If the Tree-Width was smaller than the k that was found, a better strategy for the cops and robber game would exist. This strategy, however, could then be applied in the Bi-Kelly-Width game, contradicting the original assumption.

All of the steps leading to the transfer are possible polynomial time, as we only need to convert n^2 vertices. This is even linear for a fixed k, as graphs of bounded Tree-Width and Bi-Kelly-Width are sparse. All other operations should be possible in constant or linear time.

As a result, the Bi-Kelly-Width of a graph may not be computed in polynomial time, unless P = NP. □

4. Interesting Graphs

We will now observe some interesting graphs. We will first look at general graphs of increasing Bi-Kelly-Width and bipartite graphs which are important for later constructions. Usually, the figures will contain one possible elimination order to reach the desired Bi-Kelly-Width. This order tends to also lead to the minimal Tree-Width and Kelly-Width, which are given as well.

4.1. General classes

4.1.1. Trees

Just as with Tree-Width, trees always have a Bi-Kelly-Width of one, unless they are trivial and only contain one vertex. Figure 4.1a demonstrates one possible elimination order. First, the root at the top is occupied. After that, for any occupied vertex that will be occupied, the predecessor gets blocked first. A depth-first-search would have been possible just as well.

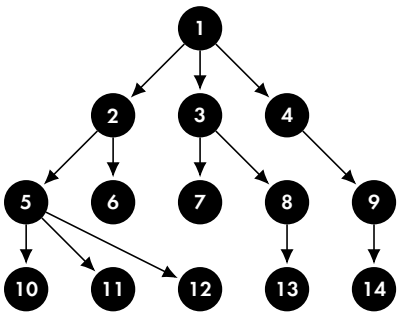
For Bi-Kelly-Width, we are able to start at any vertex for this procedure. Kelly-Width, on the other hand, may rely on a specific order if it is to take advantage of a tree being circle free and only relying on one cop. Once bi-directed edges are introduced, this ceases to be possible for Kelly-Width and it rises from zero to one (Figure 4.1b).

4.1.2. Circles

Circles always have a Tree-Width and Bi-Kelly-Width of two. Only Kelly-Width may reach lower values if the edge direction is broken and if it doesn't contain two adjacent bi-directed edges.

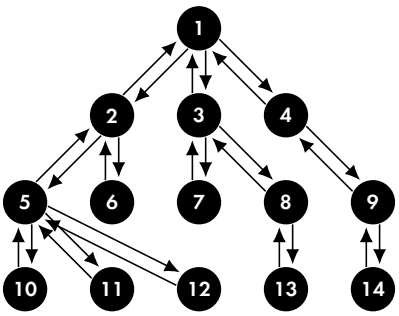
4.1.3. Complete graphs

A complete graph, even if all edges are just one-way directed as in Figure 4.1f is a haven and therefore hase Bi-Kelly-Width of $|V(G)| - 1$, similar to undirected Tree-Width. This is in contrast to Kelly-Width, which would even allow for a width of zero if there was a sink for edges.



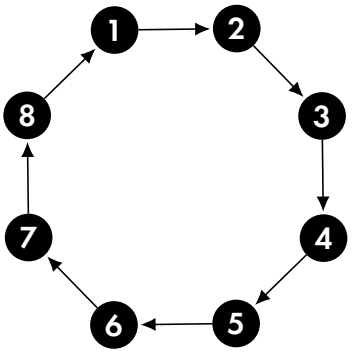
Tree-Width	1
Bi-Kelly-Width	**1**
Kelly-Width	0

(a) Ordered tree



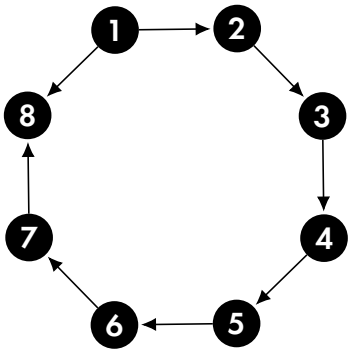
Tree-Width	1
Bi-Kelly-Width	**1**
Kelly-Width	1

(b) Full tree



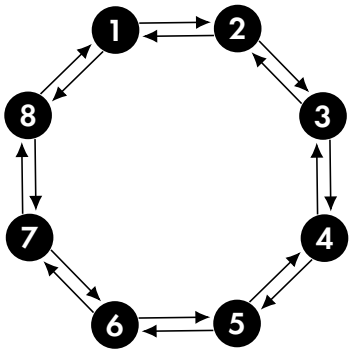
Tree-Width	2
Bi-Kelly-Width	**2**
Kelly-Width	1

(c) Ordered circle



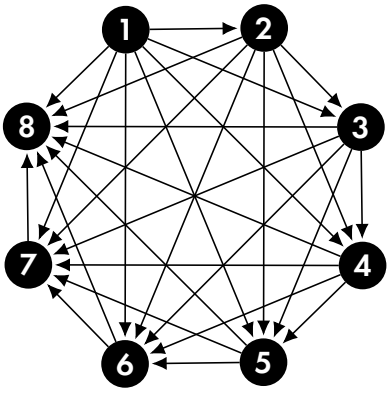
Tree-Width	2
Bi-Kelly-Width	**2**
Kelly-Width	0

(d) Broken circle



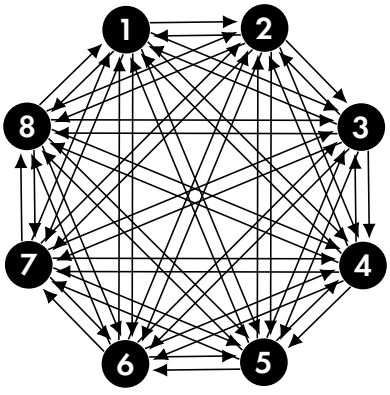
Tree-Width	2
Bi-Kelly-Width	**2**
Kelly-Width	2

(e) Full circle



Tree-Width	n-1	7
Bi-Kelly-Width	**n-1**	**7**
Kelly-Width	0	0

(f) One-way complete graph



Tree-Width	n-1	7
Bi-Kelly-Width	**n-1**	**7**
Kelly-Width	n-1	7

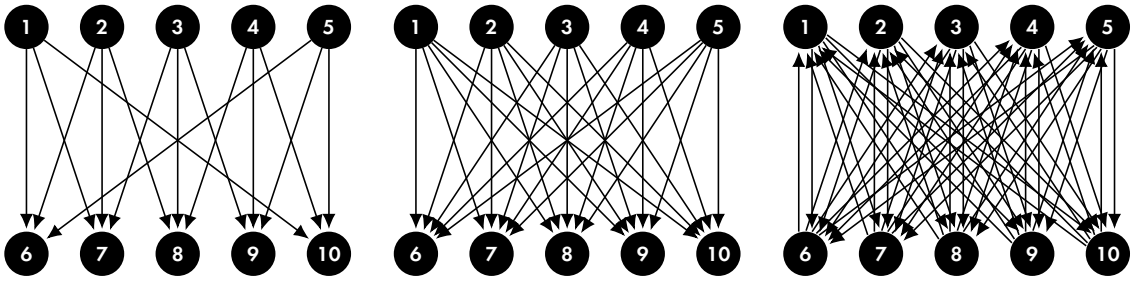
(g) Complete graph

Figure 4.1.: Basic graph classes with different edge layouts

4.1.4. Bipartite graphs

Bipartite graphs have a Tree-Width of at most $m = \frac{|V(G)|}{2}$. The cops would be placed on the smaller set of vertices (at most $\frac{|V(G)|}{2}$ for an even distribution, such as Figure 4.2c) and one remaining cop would then be able to clear all other vertices. As we can see in Figure 4.2b, this measure still needs to be applied on ordered bipartite graphs for Bi-Kelly-Width, while Kelly-Width cannot find any directed circles anymore. Furthermore, the Bi-Kelly-Width is limited by the maximum degree of either side if the edges are all directed in the same way. For example in Figure 4.2a, only four cops would suffice to clear the top row and then block all $d = 3$ possible escape routes back into the top layer.

This allows for an easy characterization of some directed graphs, for example those that either have an incoming or an outgoing degree of zero for all vertices have a Bi-Kelly-Width of at most $\frac{|V(G)|}{2}$. If the two sets of vertices do not have the same size, a strategy can take advantage of the smaller set. The same applies to having one side with a smaller maximum degree. We can also take advantage of this with some restrictions in multi-layered scenarios as can be seen in neural networks.



Tree-Width	m	5
Bi-Kelly-Width	**d**	**3**
Kelly-Width	0	0

(a) Sparse bipartite

Tree-Width	m	5
Bi-Kelly-Width	**m**	**5**
Kelly-Width	0	0

(b) Complete ordered

Tree-Width	m	5
Bi-Kelly-Width	**m**	**5**
Kelly-Width	m	5

(c) Complete bipartite graph

Figure 4.2.: Bipartite graphs with different edge layouts

4.2. Examples from literature

4.2.1. Slivkin's gadget construction

Slivkins (2010) proved W[1]-hardness for the Directed Edge Disjoint Paths Problem on acyclic directed graphs. All of these would have a Kelly-Width of zero, as they are acyclic, so Kelly-Width and Directed Tree-Width are not suitable as a parameter for a parameterized approach to this problem. But how about Bi-Kelly-Width?

At this point, we can only prove that the Bi-Kelly-Width of the acyclic graphs Slivkins constructed yields instances of arbitrarily large Bi-Kelly-Width. This does not lead to a parameterization of the Directed Edge Disjoint Paths Problem with Bi-Kelly-Width, however, it doesn't rule it out either. Slivkins used a parameterized reduction from the well-known W[1]-hard Clique Problem to the Directed Edge Disjoint Paths Problem on a specific class of acyclic directed graphs. In the Clique Problem the input is an undirected graph G and a number k and the question is whether G contains a clique of size k, that is, a set of k pairwise adjacent vertices. The reductions translate an input (G, k) for the Clique Problem to a directed graph via a quite complex construction. The full details of this construction are not relevant for us. We focus on demonstrating that the constructed instances have unbounded Bi-Kelly-Width. As we will see, there will be almost complete bipartite directed graphs as in Figure 4.2b on the facing page.

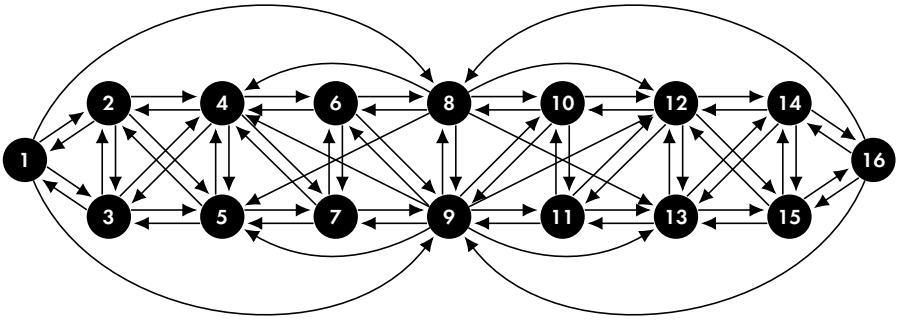
From the instance (G, k) a $k \times n$ array of identical gadgets is created. Assume that each of the n vertices of $V(G)$ exactly corresponds to some u. For each gadget $G_{i,u}$ for $1 \leq i \leq k$ and $1 \leq u \leq n$ there are k pairs of vertices $(a_1^{iu}, b_1^{iu}), \ldots, (a_k^{iu}, b_k^{iu})$. There are further vertices that serve as terminal pairs for the Directed Edge Disjoint Paths Problem as well as further complicated connections that ensure that a clique of size k exists in G if and only if the desired edge-disjoint paths exist in the constructed instance.

The part that alone creates a subgraph of unbounded Bi-Kelly-Width is the following. For each edge $\{u, v\}$ in G and each $i < j$ there is a directed edge from b_j^{iu} in G_{iu} to a_i^{jv} in G_{jv}. Assume now that the input graph G is a complete graph on n vertices and $k \geq 2$. We consider in the constructed directed graph the subgraph H with vertex set b_2^{1u} and a_1^{2v} for $u, v \leq n$. As G is complete, we have a directed edge from b_2^{1u} to a_1^{2v} if and only if $u \neq v$, hence, we have an almost complete directed bipartite graph. In fact, we have exactly $n^2 - n$ edges, hence, an edge density of $\frac{n^2-n}{2n}$, which goes to infinity as we increase our input size n.

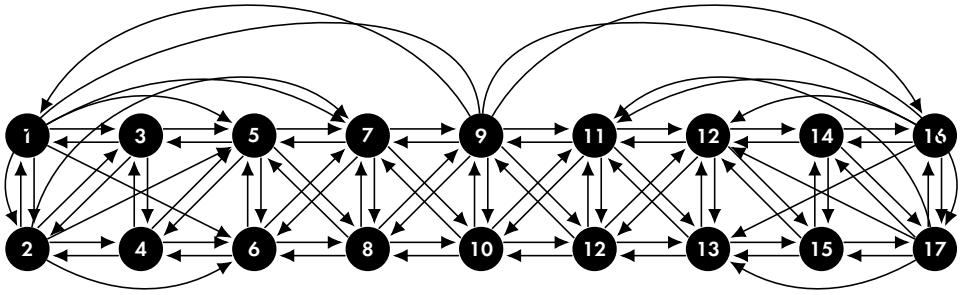
As shown in Corollary 8, Bi-Kelly-Width is closed under subgraphs. Furthermore, graph classes of bounded Bi-Kelly-Width need to be sparse by Corollary 6 on page 41. We can now conclude that the class of all instances obtained in the reduction does not have bounded Bi-Kelly-Width.

4.2.2. Adler's examples for Directed Tree-Width

Adler (2007) analyzed the properties of Directed Tree-Width with a couple of graphs. We can observe that they do result in a relatively high Bi-Kelly-Width as they are very dense. In Figure 4.3 on the following page D^1 has a Bi-Kelly-Width of 5. In contrast to Directed Tree-Width, the non-monotone strategy with one fewer cop fails. The same applies to the Bi-Kelly-Width of D^2.



(a) D^1



(b) D^2

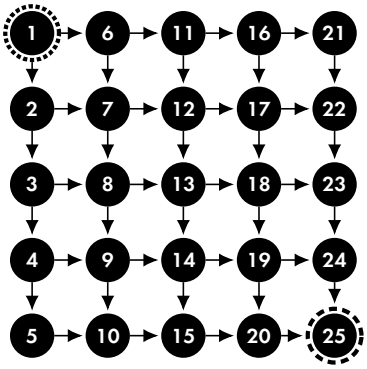
Figure 4.3.: Graphs after Adler

4.3. Complex classes

These classes of graphs usually involve larger structures and more sophisticated instructions how edges are laid out. Some cases are actually special cases of the aforementioned normal classes, especially bipartite graphs appear frequently.

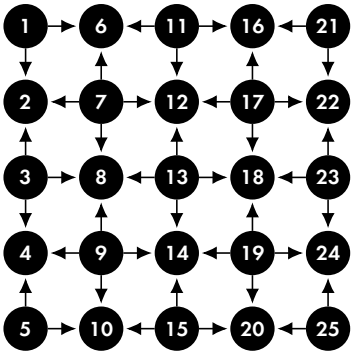
4.3.1. The grid

One of the most important classes for demonstrating the differences between the width measures are grids. For simplicity, we will only observe square grids. Otherwise, game strategies will usually take advantage of the shorter side if the width is unbounded. We will refer to m as the amount of vertices on one side, so the total amount of vertices is m^2. Tree-Width is constant with $k = m$. It therefore also serves as the upper bound for the other width measures.



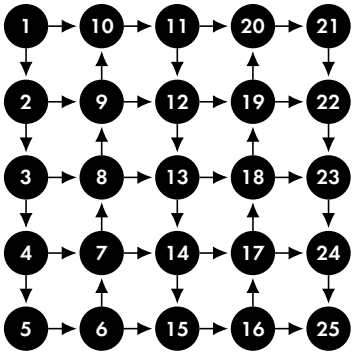
	m	5
Tree-Width		
Bi-Kelly-Width	**2**	**2**
Kelly-Width	0	0

(a) Opposing source and sink



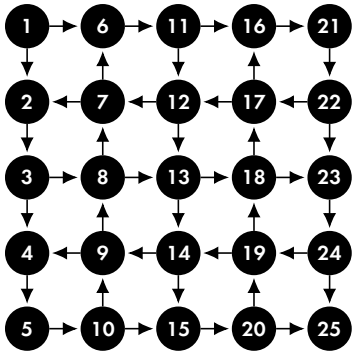
	m	5
Tree-Width		
Bi-Kelly-Width	**2**	**2**
Kelly-Width	0	0

(b) In- or out-degree of zero



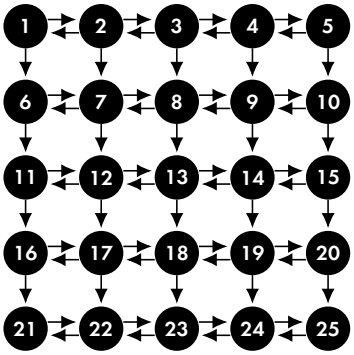
	m	5
Tree-Width		
Bi-Kelly-Width	**2**	**2**
Kelly-Width	0	0

(c) Some alternating edges



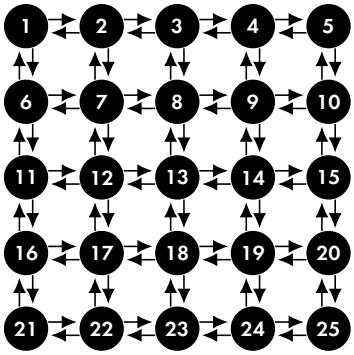
	m	5
Tree-Width		
Bi-Kelly-Width	**m/2**	**3**
Kelly-Width	m/2	3

(d) Many alternating edges



	m	5
Tree-Width		
Bi-Kelly-Width	**m**	**5**
Kelly-Width	1	1

(e) Some bi-directed edges



	m	5
Tree-Width		
Bi-Kelly-Width	**m**	**5**
Kelly-Width	m	5

(f) Full grid

Figure 4.4.: Grids with different edge layouts

In the grids in Figures 4.4a and 4.4b, the cop strategy would rely on blocking the immediate vertices leading back into already covered ground. This leads to a Bi-Kelly-Width of two, while the two cops are not required for Kelly-Width which is zero.

With only alternating edges, the grid in Figure 4.4b is actually just a directed bipartite graph and a Bi-Kelly-Width of two.

4.3.2. Cylindrical grids

Cylindrical grids can be interpreted as the outer side of a cylinder. For Tree-Width, they were among the first types of graphs being analyzed (Robertson and Seymour, 1984). Notably, they are planar graphs.

Most strategies can work on clearing the different layers of the grid one by one while securing the first vertex of a row (Figures 4.5a and 4.5b on the next page). Bi-Kelly-Width becomes unbound as soon as bi-directed edges appear, which are creating too may paths (Figure 4.5c). This however does not work once the graph becomes to dense vertically, leading to an unbounded width on all measures (Figure 4.5d).

If the cylindrical grid becomes very short with a circumference twice as large as one side, strategies can rely on blocking one vertical column and then clearing the remaining vertices like a regular grid.

4.3.3. The mesh

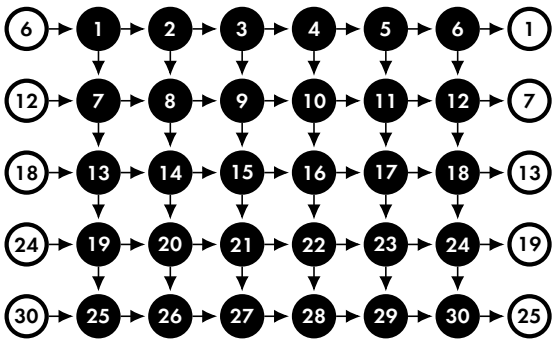
In contrast to the cylindrical grids, the mesh connects vertex in all directions. For $m > 2$, these aren't planar graphs anymore and many strategies rely on a lot of cops to block escapes through one side and effectively reduce the capturing problem to a cylindrical grid.

In some cases such as Figure 4.6a on page 58, paths may be broken by occupying a diagonal line first.

4.3.4. The hypercube

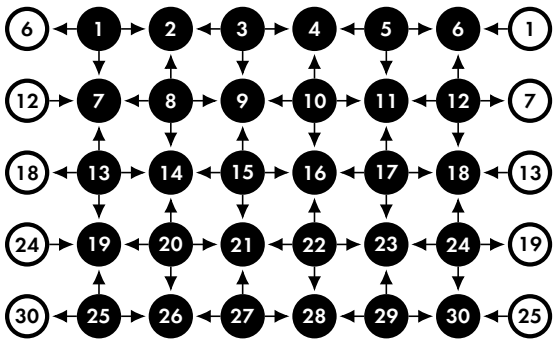
Directed m-dimensional hypercubes can have a constant Kelly-Width of zero. For Bi-Kelly-Width this value rises as the minimum degree of edges increases for each dimension.

Figure 4.7 demonstrates this with tesseracts ($m = 4$). The underlying undirected Tree-Width or the full hypercube has a width of 2^{m-1}, as all vertices of a layer need to be blocked, otherwise reoccupation would be possible. If just few paths are created (Figures 4.7a to 4.7c), Bi-Kelly-Width rises only with m.



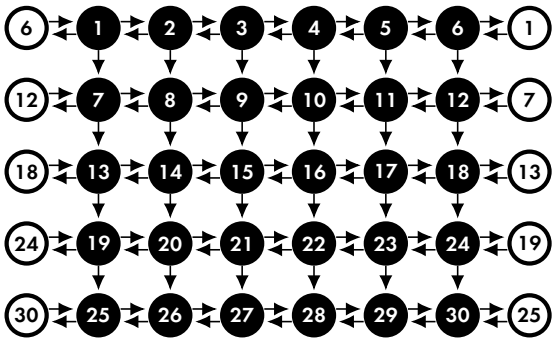
	m	6
Bi-Kelly-Width	**3**	**3**
Kelly-Width	1	1

(a) Ordered



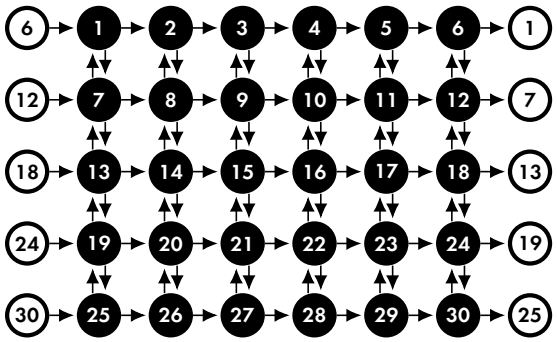
	m	6
Bi-Kelly-Width	**3**	**3**
Kelly-Width	0	0

(b) Alternating edges



	m	6
Bi-Kelly-Width	**m**	**6**
Kelly-Width	2	2

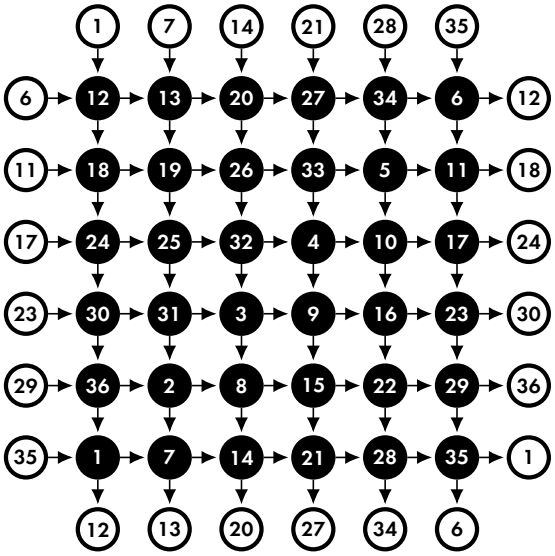
(c) Horizontally bi-directed edges



	m	6
Bi-Kelly-Width	**m**	**6**
Kelly-Width	m	6

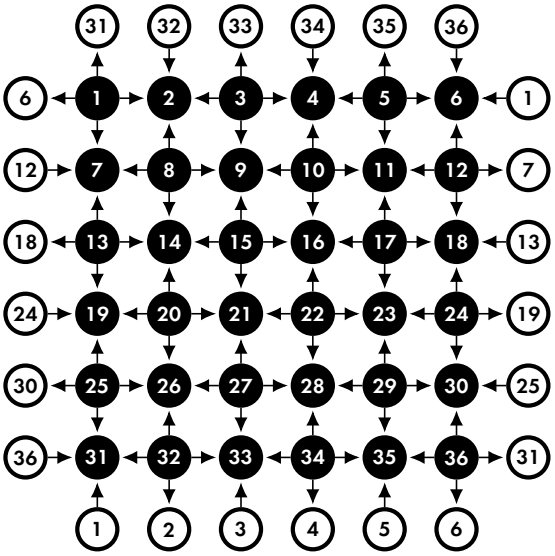
(d) Vertically bi-directed edges

Figure 4.5.: Cylindrical grids



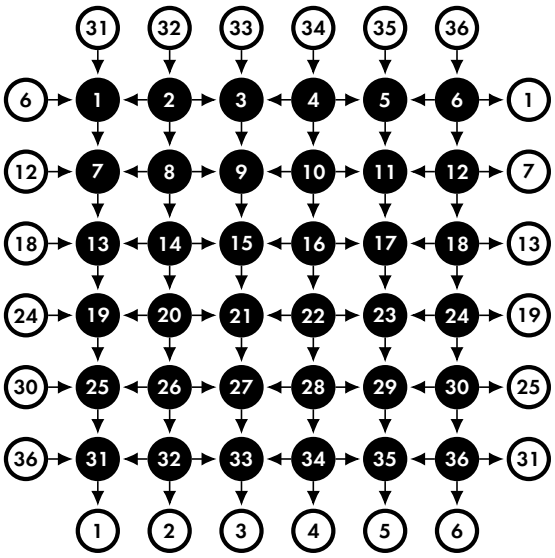
	2m	12
Tree-Width	2m	12
Bi-Kelly-Width	**m+2**	**8**
Kelly-Width	m	6

(a) Ordered



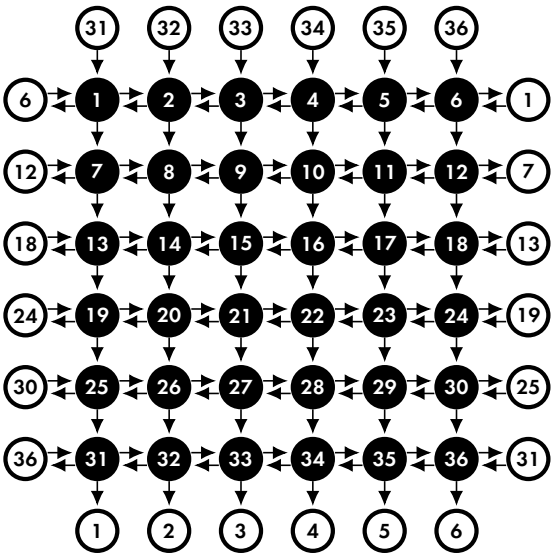
	2m	12
Tree-Width	2m	12
Bi-Kelly-Width	**4**	**4**
Kelly-Width	0	0

(b) Alternating edges



	2m	12
Tree-Width	2m	12
Bi-Kelly-Width	**6**	**6**
Kelly-Width	1	1

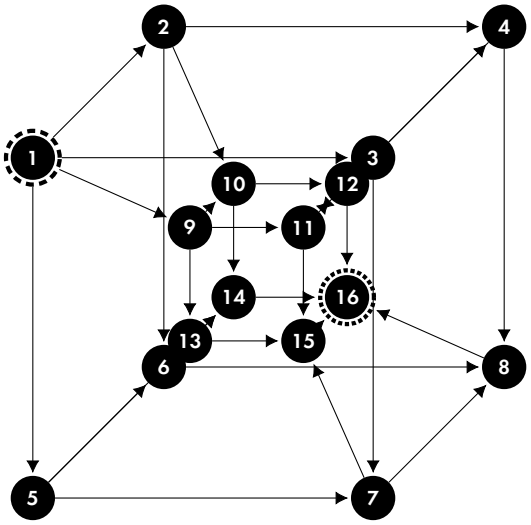
(c) Alternating rows



	2m	12
Tree-Width	2m	12
Bi-Kelly-Width	**2m**	**12**
Kelly-Width	2m	12

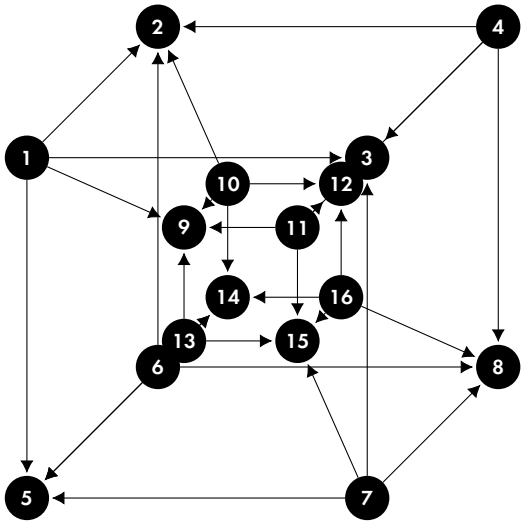
(d) Some bi-directed edges

Figure 4.6.: Different meshs



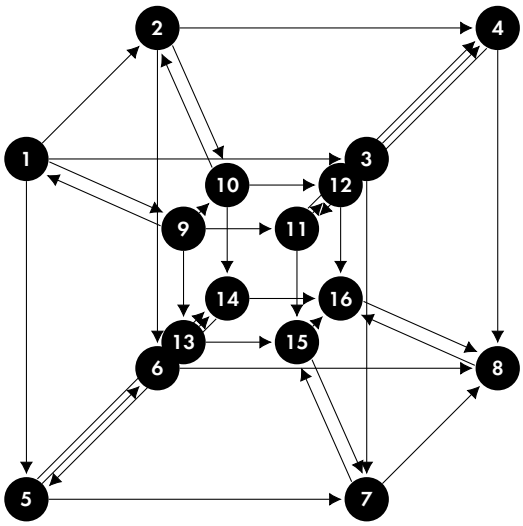
Tree-Width	2^{m-1}	8
Bi-Kelly-Width	**m**	**4**
Kelly-Width	0	0

(a) Opposing source and sink



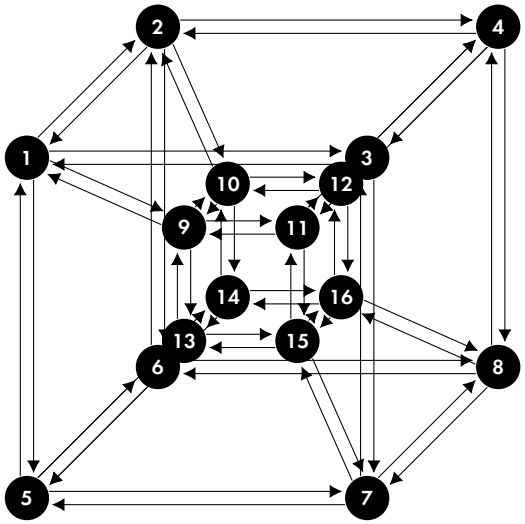
Tree-Width	2^{m-1}	8
Bi-Kelly-Width	**m**	**4**
Kelly-Width	0	0

(b) Alternating edges



Tree-Width	2^{m-1}	8
Bi-Kelly-Width	**m**	**4**
Kelly-Width	1	1

(c) Cube with some directed edges



Tree-Width	2^{m-1}	8
Bi-Kelly-Width	$\mathbf{2^{m-1}}$	**8**
Kelly-Width	2^{m-1}	8

(d) Full cube

Figure 4.7.: Different edge layouts for hypercubes

5. Conclusion

5.1. Further research

Now that the structure theory has been created, algorithmic applications need to follow. Especially finding an FPT parameterization for the Directed Edge-Disjoint Paths Problem remains an open problem.

We focused on comparing Bi-Kelly-Width with Kelly-Width and undirected Tree-Width. It could be interesting to compare Bi-Kelly-Width to other directed width measures like Directed Tree-Width (Johnson et al., 2001), DAG-Width (Berwanger et al., 2006; Obdržálek, 2006), or Kenny-Width and DAG-Depth (both Ganian, Langer, et al., 2014). This would help to understand Bi-Kelly-Width even better. Future width measures could be inspired by modifying existing directed width measures to take reversed paths into account. As it has been done with Tree-Width (Kříž and Thomas, 1991), an adoption of Bi-Kelly-Width for directed infinite graphs would seem plausible.

Considering topological minors, it would be interesting to find the minimum set of forbidden minors for graphs to have a small fixed Bi-Kelly-Width, as it has been done for Tree-Width (Arnborg, Corneil, et al., 1990) and Kelly-Width (Kintali and Zhang, 2017).

Results in computing Tree-Width, such as parameterized approaches (Bodlaender, 1992), approximation (Bodlaender, Gilbert, et al., 1992), both combined (Bodlaender, Drange, et al., 2016; Korhonen, 2021) or heuristics (Bodlaender, Fomin, et al., 2006) and integer linear programming (ILP; Bodlaender, Jansen, et al., 2011) could inspire constructions for Bi-Kelly-Width.

This would also allow for applications outside of being a parameter for complexity. It could be a useful measure for the complexity of source code or class hierarchies in software engineering. It would also be interesting to analyze the Bi-Kelly-Width of certain types of Neural Networks that do not fully connect all nodes of adjacent layers, such as Convolutional Neural Networks (LeCun and Bengio, 2002).

5.2. Summary

We have created a structure theory that nicely fits between undirected Tree-Width and Kelly-Width.

	Tree-Width	Bi-Kelly-Width	Kelly-Width
Maximum average degree / density	$2k$	$2k$	unbounded
Width of trees	1	1	0 or 1
Width of circles	2	2	0 to 2
Width of ordered grids	unbounded	2	0
Width of directed acyclic graphs	(unbounded)	unbounded	0
Width of complete graphs	n-1 (maximal)	n-1 (maximal)	may be 0
Takes edge direction into account	✗	✓	✓
Many different characterizations available	✓	✓	✓

Figure 5.1.: Comparison summary

Bi-Kelly-Width comes with various equivalent definitions. This makes it very robust. The cops and robber games example is perhaps the most intuitive and visual representation. They allow us to compare Bi-Kelly-Width with other width measures. Elimination orderings are intuitively closely related both to winning games and working in algorithmic applications. Fill-in graphs are well suited for algorithms to compute the Bi-Kelly-Width of a graph. Partial bi-directed k-trees using their generative approach are well connected to formal languages.

We used different kinds of examples to help to demonstrate its properties, most prominently grids. Even with these abstract cases it becomes apparent that we are in some cases able to find structures with bound Bi-Kelly-Width and obtain the same unbound values as undirected Tree-Width once the graphs become densely connected. This is exactly the behavior we intended, so it is well suited as a parameter for complexity.

There are many promising possible applications for Bi-Kelly-Width in several fields, and it would be possible to look into further derived concepts.

In the end, this is exactly what we have hoped to achieve. It even might be the *good* adaption of Tree-Width to undirected graphs that many researchers have been looking for.

A. Appendix

A.1. References

Adler, Isolde (2007). "Directed tree-width examples". In: *Journal of Combinatorial Theory, Series B* 97.5, pp. 718–725. DOI: 10.1016/j.jctb.2006.12.006 (cit. on pp. 53, 54).

Arnborg, Stefan, Derek G. Corneil, and Andrzej Proskurowski (1987). "Complexity of Finding Embeddings in a k-Tree". In: *SIAM Journal on Algebraic Discrete Methods*. DOI: 10.1137/0608024 (cit. on pp. 11, 23).

– (1990). "Forbidden minors characterization of partial 3-trees". In: *Discrete Mathematics* 80.1, pp. 1–19. DOI: 10.1016/0012-365X(90)90292-P (cit. on p. 60).

Arnborg, Stefan and Andrzej Proskurowski (1989). "Linear time algorithms for NP-hard problems restricted to partial k-trees". In: *Discrete Applied Mathematics* 23.1, pp. 11–24. DOI: 10.1016/0166-218X(89)90031-0 (cit. on p. 12).

Baedeker, Karl (1910). "Bremen I". In: *Handbook for Travellers*. 15th ed. New York: Charles Scribner's Sons. URL: https://commons.wikimedia.org/w/index.php?title=File:Bremen_Map_1910.jpg&oldid=481488614 (visited on 01/13/2021) (cit. on p. 8).

Bang-Jensen, Jørgen and Gregory Z Gutin (2008). *Digraphs: theory, algorithms and applications*. Springer Science & Business Media (cit. on p. 15).

Bertelé, Umberto and Francesco Brioschi (1972). *Nonserial Dynamic Programming*. Vol. 91. New York: Academic Press. DOI: 10.1016/S0076-5392(08)60141-1 (cit. on pp. 11, 18).

Berwanger, Dietmar, Anuj Dawar, Paul Hunter, and Stephan Kreutzer (2006). "DAG-Width and Parity Games". In: *23rd Annual Symposium on Theoretical Aspects of Computer Science*. Lecture Notes in Computer Science. Springer. DOI: 10.1007/11672142_43 (cit. on pp. 12, 60).

Bodlaender, Hans L. (1988). "Dynamic programming on graphs with bounded treewidth". In: *Automata, Languages and Programming*. Springer, pp. 105–118. DOI: 10.1007/3-540-19488-6_110 (cit. on p. 11).

– (1992). "A linear time algorithm for finding tree-decompositions of small treewidth". In: *Proceedings of the twenty-fifth annual ACM symposium on Theory of Computing*. Association for Computing Machinery. DOI: 10.1145/167088.167161 (cit. on p. 60).

– (1993). "A Tourist Guide through Treewidth". In: *Acta Cybernetica* 11.1-2, pp. 1–21. URL: https://cyber.bibl.u-szeged.hu/index.php/actcybern/article/view/3417 (cit. on p. 12).

Bodlaender, Hans L., Pål Grønås Drange, Markus S. Dregi, Fedor V. Fomin, Daniel Lokshtanov, and Michał Pilipczuk (2016). "A $c^k n$ 5-Approximation Algorithm for Treewidth". In: *SIAM Journal on Computing*. DOI: 10.1137/130947374 (cit. on p. 60).

Bodlaender, Hans L., Fedor V. Fomin, Arie M.C.A. Koster, Dieter Kratsch, and Dimitrios M. Thilikos (2006). "On exact algorithms for treewidth". In: *European Symposium on Algorithms*. Also published as Zuse Institute Berlin Report 06-32. DOI: 10.1007/11841036_60. URL: https://www.zib.de/de/projects/treewidth-and-combinatorial-optimization (visited on 04/18/2021) (cit. on p. 60).

Bodlaender, Hans L., John R. Gilbert, Hjálmtýr Hafsteinsson, and Ton Kloks (1992). "Approximating treewidth, pathwidth, and minimum elimination tree height". In: *Graph-Theoretic Concepts in Computer Science*. Springer, pp. 1–12. DOI: 10.1007/3-540-55121-2_1 (cit. on p. 60).

Bodlaender, Hans L., Bart M. P. Jansen, and Stefan Kratsch (2011). "Preprocessing for Treewidth: A Combinatorial Analysis through Kernelization". In: *Automata, Languages and Programming*. Springer, pp. 437–448. DOI: 10.1007/978-3-642-22006-7_37 (cit. on p. 60).

Bodlaender, Hans L. and Ton Kloks (1992). "A simple linear time algorithm for triangulating three-colored graphs". In: *STACS 92*. Springer, pp. 413–423. DOI: 10.1007/3-540-55210-3_201 (cit. on p. 12).

Breisch, Richard (1967). "An intuitive approach to speleotopology". In: *Southwestern Cavers* VI (5), pp. 72–78 (cit. on p. 20).

Courcelle, Bruno (1990). "The monadic second-order logic of graphs. I. Recognizable sets of finite graphs". In: *Information and Computation* 85.1, pp. 12–75. DOI: 10.1016/0890-5401(90)90043-H (cit. on p. 12).

Courcelle, Bruno and Joost Engelfriet (2012). *Graph Structure and Monadic Second-Order Logic: A Language-Theoretic Approach*. Encyclopedia of Mathematics and its Applications. Cambridge University Press. DOI: 10.1017/CBO9780511977619 (cit. on p. 12).

Cygan, Marek, Fedor V. Fomin, Łukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh (2016). *Parameterized Algorithms*. Springer. DOI: 10.1007/978-3-319-21275-3 (cit. on p. 11).

Diestel, Reinhard (2018). "Graph Theory". In: Graduate Texts in Mathematics. Berlin: Springer. ISBN: 9783662575604. URL: http://diestel-graph-theory.com/ (cit. on p. 18).

Downey, Rod and Michael R. Fellows (2013). *Fundamentals of Parameterized Complexity*. Springer. DOI: 10.1007/978-1-4471-5559-1 (cit. on p. 13).

Dyer, Danny (2018). "Pursuit-evasion games and visibility". Celebrating Brian Alspach's 80th and Dragan Marušič's 65th birthday. In: *Graphs, groups, and more*. Koper: Slovenian Discrete and Applied Mathematics Society. URL: https://conferences.famnit.upr.si/event/4/ (visited on 03/05/2021) (cit. on p. 20).

Ganian, Robert, Alexander Langer, Petr Hliněný, Joachim Kneis, Jan Obdržálek, and Peter Rossmanith (2014). "Digraph width measures in parameterized algorithmics". In: *Discrete Applied Mathematics* 168. Fifth Workshop on Graph Classes, Optimization, and Width Parameters, Daejeon, Korea, October 2011, pp. 88–107. DOI: 10.1016/j.dam.2013.10.038 (cit. on pp. 13, 60).

Ganian, Robert, Daniel Meister, Petr Hliněný, Joachim Kneis, Jan Obdržálek, Peter Rossmanith, and Somnath Sikdar (2016). "Are there any good digraph width measures?" In: *Journal of Combinatorial Theory, Series B* 116, pp. 250–286. DOI: 10.1016/j.jctb.2015.09.001 (cit. on pp. 13, 47).

Halin, Rudolf (1976). "S-Functions for Graphs". In: *Journal of Geometry*. DOI: 10.1007/BF01917434 (cit. on pp. 11, 18).

Hunter, Paul and Stephan Kreutzer (2008). "Digraph Measures: Kelly Decompositions, Games, and Orderings". In: *Theoretical Computer Science* 399. DOI: 10.1016/j.tcs.2008.02.038 (cit. on pp. 12, 23–27, 42).

Johnson, Thor, Neil Robertson, Paul D Seymour, and Robin Thomas (2001). "Directed Tree-Width". In: *Journal of Combinatorial Theory, Series B*. DOI: 10.1006/jctb.2000.2031 (cit. on pp. 12, 60).

Kintali, Shiva and Qiuyi Zhang (2017). "Forbidden directed minors and Kelly-width". In: *Theoretical Computer Science* 662, pp. 40–47. DOI: 10.1016/j.tcs.2016.12.008 (cit. on p. 60).

Kirousis, Lefteris M. and Christos H. Papadimitriou (1986). "Searching and pebbling". In: *Theoretical Computer Science* 47, pp. 205–218. DOI: 10.1016/0304-3975(86)90146-5 (cit. on p. 11).

Korhonen, Tuukka (2021). *Single-Exponential Time 2-Approximation Algorithm for Treewidth*. arXiv: 2104.07463 [cs.DS] (cit. on p. 60).

Kreowski, Hans-Jörg, Renate Klempien-Hinrichs, and Sabine Kuske (2006). "Some essentials of graph transformation". In: *Recent Advances in Formal Languages and Applications*. Studies in Computational Intelligence. Heidelberg: Springer, pp. 229–254. DOI: 10.1007/978-3-540-33461-3_9 (cit. on p. 17).

Kříž, Igor and Robin Thomas (1991). "The menger-like property of the tree-width of infinite graphs". In: *Journal of Combinatorial Theory*. B. DOI: 10.1016/0095-8956(91)90093-Y (cit. on p. 60).

LaPaugh, Andrea S. (1993). "Recontamination does not help to search a graph". In: *Journal of the ACM* (2). DOI: 10.1145/151261.151263 (cit. on pp. 11, 21).

LeCun, Yann and Yoshua Bengio (2002). *Convolutional Networks for Images, Speech, and Time Series*. Ed. by Michael A. Arbib. 2nd ed. Cambridge, MA, USA: MIT Press, pp. 255–258. DOI: 10.7551/mitpress/3413.003.0006 (cit. on p. 60).

Makowsky, Johann A. and Julian P. Mariño (2003). "Tree-width and the monadic quantifier hierarchy". In: *Theoretical Computer Science* 303.1. Logic and Complexity in Computer Science, pp. 157–170. DOI: 10.1016/S0304-3975(02)00449-8 (cit. on p. 12).

Möhring, Rolf H. (1990). "Graph Problems Related to Gate Matrix Layout and PLA Folding". In: *Computational Graph Theory*. Vienna: Springer Vienna, pp. 17–51. DOI: 10.1007/978-3-7091-9076-0_2 (cit. on p. 12).

Nešetřil, Jaroslav and Patrice Ossona de Mendez (2012). *Sparsity. Graphs, Structures, and Algorithms*. Springer. DOI: 10.1007/978-3-642-27875-4 (cit. on p. 22).

Obdržálek, Jan (2006). "DAG-width. Connectivity measure for directed graphs". In: *Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm*. Symposium on Discrete Algorithms SODA'06. DOI: 10.1145/1109557.1109647 (cit. on pp. 12, 60).

Reed, Bruce A. (1999). "Introducing Directed Tree Width". In: *Electronic Notes in Discrete Mathematics* 3. 6th Twente Workshop on Graphs and Combinatorial Optimization, pp. 222–229. ISSN: 1571-0653. DOI: 10.1016/S1571-0653(05)80061-7 (cit. on p. 12).

Robertson, Neil and Paul D. Seymour (1984). "Graph minors. III. Planar tree-width". In: *Journal of Combinatorial Theory, Series B*. DOI: 10.1016/0095-8956(84)90013-3 (cit. on pp. 11, 18, 19, 56).

– (1995). "Graph Minors .XIII. The Disjoint Paths Problem". In: *Journal of Combinatorial Theory, Series B* 63.1, pp. 65–110. DOI: 10.1006/jctb.1995.1006 (cit. on p. 12).

– (2004). "Graph Minors. XX. Wagner's conjecture". In: *Journal of Combinatorial Theory, Series B* 92.2, pp. 325–357. DOI: 10.1016/j.jctb.2004.08.001 (cit. on p. 18).

Rose, Donald J. (1970). "Triangulated graphs and the elimination process". In: *Journal of Mathematical Analysis and Applications*. DOI: 10.1016/0022-247X(70)90282-9 (cit. on p. 22).

Schneidewind, Uwe, Stefan Lechtenböhmer, Christa Liedke, Stefan Thomas, Henning Wilts, Carolin Baedeker, Christiane Beuermann, Ralf Schüle, and Peter Viebahn (2018). *Die große Transformation. Eine Einführung in die Kunst gesellschaftlichen Wandels*. 1st ed. Frankfurt am Main: Fischer Taschenbuch. ISBN: 978-3-596-70259-6 (cit. on p. 7).

Sedgewick, Robert and Kevin Wayne (2007). *Directed Graphs*. URL: https://www.cs.princeton.edu/courses/archive/spring07/cos226/lectures.html (visited on 04/18/2021) (cit. on p. 7).

Seymour, Paul D. and Robin Thomas (1993). "Graph Searching and a Min-Max Theorem for Tree-Width". In: *Journal of Combinatorial Theory*. DOI: 10.1006/jctb.1993.1027 (cit. on pp. 11, 21).

Slivkins, Aleksandrs (2010). "Parameterized Tractability of Edge-Disjoint Paths on Directed Acyclic Graphs". In: *SIAM Journal on Discrete Mathematics*. DOI: 10.1137/070697781 (cit. on pp. 13, 52, 53).

All references have been verified on May 14, 2021. Digital Object Identifiers (DOIs) can be resolved through https://www.doi.org/.

A.2. Glossary

Bipartite graph A graph with connections solely between two sets, members of each set are not connected. **17**, 40, 44, 50, 52–55

Clique A completely connected component of a graph. **17**, 22, 40, 53

Complexity theory Branch of computer science that analyzes the resource cost of algorithms. 11

DAG Directed acyclic graph. 24, 27, see Digraph

DAG-Width A width measure on directed graphs. 12, 47, 60

Digraph A graph with directed edges. 10, 13, **15**, 24–27, 47

Directed Edge Disjoint Paths Problem A problem on directed graphs. 10, 12, 13, 52, 53

Directed Tree-Width A width measure on directed graphs. 12, 13, 52, 53, 60

FPT Fixed parameter tractable, $W[0]$ in the W-hierarchy. 11, 13, 60, see Parameterized complexity

k-DAG A graph similar to a DAG. Smaller k is more similar, a 0-DAG is a DAG. 24, 26–28, 38, 68, see DAG

Kelly-Width A width measure on directed graphs. 12–14, **23**, 24–27, 29, 34, 36, 38, 42, 47, 50–52, 55–61

NP Nondeterministic polynomial time; complexity class. 9–13, 23, 48, 49, see Complexity theory

P Polynomial time; complexity class. see Complexity theory

Parameterized complexity Branch of complexity theory that distinguishes between simple and easy instances for algorithms. 13

Tree-Width A width measure on undirected graphs. 11–14, 18–24, 40, 41, 47, 48, 50–52, 54–61, 68, 69

W[1] part of the W-hierarchy. 12, 13, 52, 53, see Parameterized complexity

A.3. Symbols and naming conventions

$O(x)$ Worst case running time depending on input size n ignoring constant factors

$\vee$ Logical or

$\wedge$ Logical and

$\emptyset$ Empty set

$G = (V, E)$ Graph with vertices V and edges E

F Fill-in graph

T Tree

$V(G)$ Vertices of a graph G, typically u, v, w

$E(G)$ Edges of graph G, typically $e = (u, v)$

$u \rightsquigarrow v$ Path connecting u and v

$\leq$ A linear order on elements of a set $(v_1, \ldots, v_n)$

$A \subseteq B$ Subset or, when on graphs, subgraph

$A \cup B$ Set (or graph) union

$A \cap B$ Set (or graph) intersection

$A \setminus B$ All elements of set A that are not in set B

$|X|$ Cardinality, or number of elements of a set

$G[\leq v]$ Subgraph of G with vertices smaller than or equal v with respect to $\leq$

$V[\leq v]$ Subset of V with vertices only smaller than or equal v with respect to $\leq$

$\mathrm{Reach}^{\rightarrow}(G, W, v)$ Vertices in G reachable with a path over vertices of W from v

$\mathrm{Reach}^{\leftrightarrows}(G, W, v)$ Vertices in G reachable with a path from v over vertices of W or vice-versa

$\mathrm{Reach}(G, W, v)$ Reachable vertices on undirected graphs

$\mathrm{SReach}^{\rightarrow}(G, \leq, v)$, $\mathrm{SReach}^{\leftrightarrows}(G, \leq, v)$ Strongly reachable vertices according to $\leq$

$GG = (\Sigma, S, P, T)$ Graph grammar with alphabet Σ, initial graph S, rules P and terminals T

A.4. List of figures

Figure 3.7g on page 41 also appears on the title page.

A.5. Acknowledgements

Writing this thesis would not be possible without my family, friends, and the people I work with. Keeping in touch and staying together, especially during these dire times of a pandemic, is a privilege.

I am grateful for the support I have received by Sebastian and his group, Sabine and my fellow students. Thank you, Jona, Leo and Marc for your valuable feedback!

I would also like to thank the contributors to open source projects and communities such as LaTeX, all of its packages and extensions, GIMP and GitLab, that I have relied on while creating this document.

Furthermore, the Wikipedia article on Tree-Width provides an excellent introduction to its topic, though I have primarily used other sources as references.

Statutory declaration

I, Enna Gerhard, declare that I have produced this thesis without outside assistance. I have not used other sources or means than declared. All parts that are reproducing or paraphrasing sources have been attributed as such.

Bremen, May 14, 2021

_____

(Enna Gerhard)