Technische Universität München
TUM Campus Straubing für Biotechnologie und Nachhaltigkeit

Algorithms for Municipal Flood Mitigation, Related Interdiction Problems, and Knapsack Problems

Jan Boeckmann

Vollständiger Abdruck der vom TUM Campus Straubing für Biotechnologie und Nachhaltigkeit der Technischen Universität München zur Erlangung eines Doktors der Naturwissenschaften (Dr. rer. nat.) genehmigten Dissertation.

Vorsitz: Prof. Dr. Alexander Hübner

Prüfer:innen der Dissertation:

1. Prof. Dr. Clemens Thielen
2. Prof. Dr. Andreas Wiese

Die Dissertation wurde am 30.05.2023 bei der Technischen Universität München eingereicht und durch den TUM Campus Straubing für Biotechnologie und Nachhaltigkeit am 06.10.2023 angenommen.

Abstract

Kenia 2020, Central Europe 2021, Pakistan 2022 – as a consequence of climate change, pluvial flash flood disasters have been increasing both in their intensity and their frequency and will continue to do so within the next years. A significant number of regions with mostly stable weather conditions until the 21st century have meanwhile witnessed exceptional pluvial flash flood disasters. Therefore, the design of flood mitigation concepts has become one of the most critical topics in settlement development, even for small municipalities. Although the design of mitigation concepts involves a complex decision problem, only simulations are typically used as a digital decision support.

This work starts with presenting the results of the project AKUT – an acronym for the German translation of "Incentive Systems for Municipal Flood Prevention" – where a decision support tool was developed that computes best-possible combinations of precautionary measures for pluvial flash floods subject to a budget constraint and taking the cooperation of local residents into account. The emerging software has meanwhile been used by over 30 institutions such as municipalities, engineering offices, and academic research groups.

Afterwards, the closely related network flow interdiction problem is investigated and an approximation algorithm is presented. This problem is known to be hard to approximate and the presented algorithm is the first whose approximation ratio does not depend on the size of the input graph. Further, on the class of simple graphs, the approximation ratio dominates the one of the previously best known approximation algorithm.

Subsequently, the complexity of several versions of the shortest path interdiction problem is investigated on temporal graphs, which have lately attracted the interest of the research community due to their high applicability to real-world problems. Four different notions of the term "shortest" in the temporal setting are introduced and the corresponding shortest path interdiction problems are investigated. Although the shortest path interdiction problem is $\mathcal{NP}$-hard on static graphs, it is found that two of the four versions are polynomial-time solvable while the other two are $\mathcal{NP}$-hard. On extension-parallel temporal graphs, however, the two hard versions are also polynomial-time solvable. Further, the complexities for three extensions of the problem are assessed.

Finally, a highly generalized version of the well-studied knapsack problem, where weights and profits are non-linear and the operator in the objective function can be a sum or a product, is studied and an approximation algorithm is presented. This algorithm is levered to the multi-objective version of the problem and the approximation algorithm for the multi-objective problem is applied to obtain further approximation algorithms for the recently introduced 0-1 time-bomb knapsack problem and some min-max and max-min versions motivated from the field of robust optimization.

Acknowledgements

First of all, I want to thank Clemens Thielen for his unconditional support and his strong commitment as a supervisor. By his extraordinary guidance and his ability to give feedback on an exceptionally fine level I could enrich my abilities both on a technical and a personal level. This work would not have been possible without his suggestions and improvements.

Also, I want to thank the whole Optimization Research Group Kaiserslautern for their unique sense of humor, the competitive games at the table soccer, and their social events. The working atmosphere was the most enjoyable I ever experienced. Special thanks go to Oliver Bachtler for his advice when I needed some and with whom sharing an office left nothing to ask for and to Luca Schäfer, Marco Natale, and Till Heller for our sports sessions and our online-meetings during the pandemic.

Further, I want to thank the whole AKUT project team for their commitment and our fruitful discussions and workshops. Special thanks go to Mirjam Lawens, who put extensive work into the software and its management and with whom I had numerous, sometimes more sometimes less sophisticated but always enjoyable discussions. Also, I want to thank Florian Meier for taking on the technical support for the software AKUT after my time at TUM Campus Straubing and for his pleasant and goal-oriented way of working.

I thank Ulrich Pferschy for being my mentor as a PhD candidate and for our mutual research stays where I met him as a very agreeable and supportive person.

Further, I want to thank Alexandra Hering and Marina Zapilko for their efforts in protecting me from administrative work. A huge thank goes to the whole first floor of Essigberg for their company during lunch breaks and at the Gäubodenvolksfest.

I would like to thank my housemates for their mental support during the pandemic, for the evenings spent in the kitchen, and for their support in organizing triathlons in front of the door when regular racing was not possible.

I would especially like to thank the afforementioned Luca Schäfer, Marco Natale, Till Heller, Oliver Bachtler, Florian Meier, as well as Arne Herzel, Johannes Kager, and Rüdiger Boeckmann for proofreading parts of this thesis.

Very special thanks go to Lena Riesenegger for her mental support both as a colleague and as a friend and partner in Straubing. Finally, the greatest thanks are dedicated to my family for their support, which ultimately paved this path for me.

Contents

1 Introduction

In this introductory chapter, we provide a motivation for the problems studied in this thesis as well as a brief outline of the content and a list of publications this thesis is based on.

1.1 Motivation

Flash floods have been increasingly affecting the worldwide news in the last years [FS21; Gua22; New20] and scientists agree that dangerous weather events in general and particularly heavy rainfall events are becoming more frequent and severe as a consequence of climate change [Ble+18; IPC21; RS17]. While there is clear evidence of the efficacy of precautionary measures [Kre+05], the design of flood mitigation concepts in Germany is typically based on simulations rather than using optimization methods [Ger16], clearly lacking the consideration of site-specific interplay of actions. This is also pointed out in [Tas21; WKG14] for the design of flood mitigation concepts and the sparse literature on the topic indicates that optimization techniques are globally rarely used in the design of precautionary measures for pluvial flash floods. Regarding the urgency of the problem of adapting to heavier rain events and their tremendous potential for damage, the potential of using optimization techniques in the design of mitigation concepts is evident.

Since the quality of mitigation concepts is usually assessed by simulating flows on a digital terrain model, which could be interpreted as a directed graph, it is a natural question by how much the maximum inflow into a node can be reduced by removing a limited number of arcs from the graph, which intuitively corresponds to, e.g., building a ditch or embankment. This problem is known as the network flow interdiction problem and has been extensively studied in the literature [Bur+03; CZ17; Woo93]. Apart from its already mentioned applicability to flood mitigation, other applications include highway transportation [Dur66] and the combat of criminal drug smuggling networks [MRS12].

Various versions of interdiction problems on static graphs have been studied in the literature. However, the assumption that the graph does not change over time is quite restrictive when approaching real-world problems. Temporal graphs – a more general concept of graphs that allows for a temporal presence of connections – have recently been attracting the research community. Despite their great applicability to real world problems, the literature on interdiction problems on temporal graphs is surprisingly sparse leaving interesting problems to investigate.

Finally, one of the most essential problems in combinatorial optimization is the knapsack problem, where items have to be packed into a knapsack such that their total weight does not exceed the knapsack capacity while maximizing the total profit of the packed items. Despite its quite restrictive assumption of separable profit and weight functions, there is a considerable amount of real-world applications of the problem and new versions are still investigated [MPS22]. While there exist approximation algorithms for most studied versions of the knapsack problem, an approximation algorithm that works for the vast majority of knapsack problems marks an important gap in the literature.

1.2 Outline

This thesis consists of seven chapters including this introductory chapter, in which a motivation and overview of the content is provided.

In Chapter 2, although the reader is assumed to have basic mathematical knowledge, essential concepts of discrete optimization, complexity theory, graphs and networks, multicriteria optimization, and approximation algorithms are presented.

A mixed-integer programming approach to finding best-possible combinations of precautionary measures for pluvial flash floods and a corresponding web application are discussed in Chapter 3. We start by formally introducing the problem and presenting the input data, which are available to German municipalities free of charge. Next, we present a combinatorial algorithm, which, given a set of precautionary measures, computes the water levels in a rain event, and a mixed-integer programming formulation of the problem of finding an optimal combination of such measures subject to a budget constraint and the cooperation of local residents. The work is continued by proving that the presented mixed-integer programming formulation is indeed a valid formulation of the problem and computational results are presented, where the results are validated by comparing them to the results of state-of-the-art simulation software and the most important drivers for the quality of the obtained solution and the running time of the algorithm are pointed out.

In Chapter 4, an approximation algorithm for the special case of the network flow interdiction problem, where a limited number of arcs are to be removed from a network such that the value of a maximum s-t-flow in the resulting network is minimized, is presented. To the best of our knowledge, this is the first approximation algorithm for any version of the network flow interdiction problem whose approximation ratio does not depend on the size of the network and, on simple graphs, its approximation ratio dominates the one of the previously best known approximation algorithm.

Afterwards, a complexity analysis of several versions of the shortest path interdiction problem on temporal graphs is presented in Chapter 5. We start by introducing four definitions of the term "shortest" on temporal graphs and investigate the complexity of the four resulting versions of the shortest path interdiction problem. Interestingly, although the shortest path interdiction problem is $\mathcal{NP}$-hard on static graphs, two of the four versions are polynomial time solvable while the other two are $\mathcal{NP}$-hard on temporal graphs. On extension-parallel temporal graphs, however, the two hard versions are polynomial-time solvable. Subsequently, we show how our results generalize if negative traversal times are allowed. We then continue by introducing a more general class of temporal graphs which allows for continuous time availability of arcs and show that even deciding whether two nodes can be separated by removing a fixed number of arcs is $\mathcal{NP}$-hard. Finally, we show that our previous complexity results remain valid if we additionally impose a constraint on the maximal waiting time in a node.

Next, we present a fully polynomial-time approximation scheme (FPTAS) for a highly generalized version of the knapsack problem in Chapter 6, where profits and weights are non-linear and the operator in the objective function can be a sum or a product. The generality of this version allows to apply this algorithm to almost all versions of the knapsack problem in the literature. We then extend the idea of the FPTAS to the multiobjective case to obtain a multiobjective fully polynomial-time approximation scheme (MFPTAS) and further show how the algorithms can be modified to obtain an FPTAS or MFPTAS for the minimization knapsack problem. Finally, we use the proposed algorithms to obtain the first FPTAS for the recently-introduced 0-1 time-bomb knapsack problem and for some max-min or min-max versions motivated by the field of robust optimization.

Although Chapters 3-6 are concluded individually, we summarize the content of the thesis and conclude this work from a more holistic viewpoint in Chapter 7.

1.3 Contributions and Credits

All results in this thesis have been jointly developed under the supervision of Prof. Dr. Clemens Thielen and most of the results have been published in journal articles or conference proceedings.

A manuscript with the contents of Chapter 3 is currently in the revision process:

> J. Boeckmann and C. Thielen. *New ways in municipal flood mitigation: A mixed-integer programming approach and its practical application.* Under revision at Operations Research Forum. 2023

Chapter 4 is based on the publication

> J. Boeckmann and C. Thielen. "A (B+1)-approximation for network flow interdiction with unit costs". In: *Discrete Applied Mathematics* (2021), 1–13. DOI: 10.1016/j.dam.2021.07.008,

which itself is a generalized version of the following previous publication:

> J. Boeckmann and C. Thielen. "An Approximation Algorithm for Network Flow Interdiction with Unit Costs and Two Capacities". In: *Proceedings of the 18th Cologne-Twente Workshop on Graphs and Combinatorial Optimization (CTW).* 2020, 157–169. DOI: 10.1007/978-3-030-63072-0_13

The content of Chapter 5 is accepted for publication in the proceedings of the Second Symposium on Algorithmic Foundations of Dynamic Networks (SAND 2023).

> J. Boeckmann, C. Thielen, and A. Wittmann. *Complexity of the Temporal Shortest Path Interdiction Problem.* Accepted for the Proceedings of the 2nd Symposium on Algorithmic Foundations of Dynamic Networks (SAND 2023). 2023

Finally, Chapter 6 is based on the following journal article:

> J. Boeckmann, C. Thielen, and U. Pferschy. "Approximating single-and multi-objective nonlinear sum and product knapsack problems". In: *Discrete Optimization* 48 (2023), 100771. DOI: 10.1016/j.disopt.2023.100771

The works of Chapters 3 and 4 have been partially supported by the German Federal Ministry for the Environment, Nature Conservation and Nuclear Safety (BMU) within the project "AKUT – Incentive Systems for Municipal Flood Prevention" (grant number 67DAS156C).

2 Preliminaries

In this chapter, we present the most important definitions and concepts that are used throughout this thesis. However, since the knowledge of basic mathematical concepts in discrete optimization is assumed, we refrain from introducing the concepts in detail and refer to suitable literature at the beginning of each section instead.

2.1 Optimization Problems

Optimization problems are encountered in various forms and complexities in everyday life. Whether you run a company and try to operate your machines such that the profit is maximized (which might be a very challenging task), or you simply try to find the best bus that gets you to work on time, you have encountered an optimization problem. In this section, we present some important classes of optimization problems. For a more detailed overview, we refer to [PR14; Sch98].

Linear programs Given a set $\mathcal{X}$ of feasible solutions, the aim in an *instance* of an optimization problem is to find the solution that maximizes or minimizes the *objective function* $\alpha : \mathcal{X} \to \mathbb{R}$. The optimization problem is written as

$$\begin{aligned} \text{max or min} \quad & \alpha(x) \\ \text{subject to (s.t.)} \quad & x \in \mathcal{X}. \end{aligned}$$

An *optimization problem* is given by the set of its instances. A particularly important class of optimization problems are *linear optimization problems* or *linear programs* (LPs), which are the optimization problems whose instances can be expressed of the form

$$\begin{aligned} \text{max or min} \quad & c^T x \\ \text{s.t.} \quad & Ax \leqslant b \\ & x \in \mathbb{R}^n, \end{aligned}$$

where $c \in \mathbb{R}^n$, $b \in \mathbb{R}^m$, and $A \in \mathbb{R}^{m \times n}$. Numerous real-world optimization problems can be expressed as linear programs and there exist highly efficient algorithms that solve extremely large instances within seconds on a regular computer. A well-studied LP is the *continuous knapsack problem*, where one is given a knapsack capacity $C \in \mathbb{R}$ and a set of items, each assigned a weight $w_i \in \mathbb{R}$ and a profit $p_i \in \mathbb{R}$. The aim in the continuous knapsack problem is to (possibly partially) pack items into the knapsack such that the total weight of (partially) packed items does not exceed the knapsack capacity and the sum of the profits of (partially) packed items is maximized. The problem can be expressed as

$$\begin{aligned} \max \quad & \sum_{i=1}^{n} p_i \cdot x_i \\ \text{s.t.} \quad & \sum_{i=1}^{n} w_i \cdot x_i \leqslant C \\ & 0 \leqslant x_i \leqslant 1 \qquad \forall i \in \{1, \dots, n\} \\ & x \in \mathbb{R}^n. \end{aligned}$$

Mixed integer programs The continuous knapsack problem, however, becomes significantly more difficult as soon as we do not allow the items to be packed partially anymore, i.e., if the x_i can be either zero or one. This yields the well-studied *0-1 knapsack problem*, which is revised in further detail in Chapter 6. By imposing an additional integrality constraint on some, or possibly all, of the variables of an LP, we obtain the class of *mixed integer optimization problems* or *mixed integer programs* (MIPs), often also referred to as *mixed integer linear programs.* These problems are significantly harder to solve in general and some specific ones have challenged researchers for decades.

All problems in this work belong to the class of mixed-integer programs, which are, in general, hard to solve. Even small instances with just a few hundred variables can often not be solved to optimality by the most refined state-of-the-art solvers in reasonable time.

2.2 Complexity

In this section, we provide an overview of the most important concepts of complexity theory. The overview is based on [GJ79], which is recommended for a more detailed explanation.

Encoding schemes An *alphabet* Σ is a non-empty finite set of symbols and Σ^* is the set of all finite strings of symbols in Σ. A set $L \subseteq \Sigma^*$ is called a *language* over the alphabet Σ. For each $s \in L$, we denote by $|s|$ the *encoding length* of s, i.e. the number of symbols in the string s. An *encoding scheme* is a function that maps numbers or problem instances to strings over the alphabet Σ.

In this thesis, we assume that $\Sigma = \{0, 1\}$ and speak of *bits* instead of symbols. Further, we assume a "reasonable" encoding of the numbers and instances (see also [GJ79]), which means that an integer $n \in \mathbb{Z}$ is encoded binary with encoding length $\lceil \log_2(n) \rceil + 1$ bits ($\lceil \log_2(n) \rceil$ for the absolute value and one for the sign). Further a rational number $p/q \in \mathbb{Q}$ is encoded by encoding the two integers p and q. This motivates the convention that $\log$ denotes the logarithm to the base two. As a final remark, we will refrain from encoding irrational numbers, since they cannot be encoded in finite encoding length without rounding appropriately. For the interested reader, a different kind of complexity theory that also allows for irrational numbers is presented in [Blu+98].

Asymptotic analysis The *running time* of an algorithm on an input instance is the number of execution steps the algorithm needs on the instance until it terminates. For a more detailed way of expressing the running time of an algorithm, we refer to [Aus+12]. In the analysis of algorithms, one is typically interested in the running time of the algorithm compared to the encoding length (often also called *size*) of the input. However, since particularly running times for large problem instances are of interest, it is common to express the running time of an algorithm in the $\mathcal{O}$-notation, which only accounts for the most dominant terms of the running time as the size of the input tends to infinity. Formally, given $n_1, n_2 \in \mathbb{N}_0$, $X \subseteq \mathbb{Z}^{n_1} \times \mathbb{R}^{n_2}$, and two functions $f, g : X \to \mathbb{R}$, we say that $f(x) \in \mathcal{O}(g(x))$ if there exist constants $b, c > 0$ and some $\bar{x} \in X$ such that $f(x) \leqslant c \cdot g(x) + b$ for all $x \geqslant \bar{x}$ (which means that $x_i \geqslant \bar{x}_i$ for all $i \in \{1, \ldots, n_1 + n_2\}$). In Chapter 6 of this thesis, we analyze an algorithm, whose running time depends on $1/\varepsilon$ for some variable $\varepsilon > 0$. For convenience, we introduce the convention that, whenever a variable denoted by ε appears in the argument of the $\mathcal{O}$-notation, it tends to zero from above. We say that an algorithm has *polynomial running time* or is a *polynomial-time algorithm* if, for each instance, there exists a polynomial g such that the running time of the algorithm on the instance is in $\mathcal{O}(g(x))$, where x is the encoding length of the instance.

Decision problems A problem P is called a *decision problem* if its set of instances can be partitioned into a set Y_P of *yes-instances* and a set N_P of *no-instances* and the task is to decide to which of the two sets a given problem instance belongs. An algorithm *solves* a decision problem if it halts and assigns each given instance correctly. What makes decision problems so important in combinatorial optimization is the fact that every optimization problem has an associated decision problem by asking whether, given a value $K \in \mathbb{R}$, there exists a solution with objective value K or better. Solving the optimization problem to optimality clearly solves the associated decision problem for any given K. Reversely, if the associated decision problem can be solved efficiently, one can solve the optimization problem by applying a binary search on K.

Complexity classes While there exists a variety of complexity classes, we define the two most important classes in complexity theory, which are $\mathcal{P}$ and $\mathcal{NP}$. The class $\mathcal{P}$ consists of the decision problems for which there exists a deterministic polynomial-time algorithm. The class $\mathcal{NP}$ consists of the decision problems for which there exists a non-deterministic polynomial-time algorithm, i.e., for each yes-instance, there exists a certificate s whose encoding length is polynomial in the encoding length of the problem instance and a polynomial-time algorithm that verifies the yes-instance.

While it is clear that $\mathcal{P} \subseteq \mathcal{NP}$, the question whether $\mathcal{P} \neq \mathcal{NP}$, which is assumed by the research community, is one of the yet unsolved famous millennium problems and, therefore, one of the most challenging open problems in mathematics and computer science.

Reducibility and Hardness A decision problem P_1 is called *(polynomial-time) reducible* to a decision problem P_2 if there exists a polynomial-time algorithm that transforms any instance x_1 of P_1 into an instance x_2 of P_2 such that x_1 is a yes-instance of P_1 if and only if x_2 is a yes-instance of P_2. In this case, we write $P_1 \propto P_2$. Informally speaking, this means that, if P_2 can be solved in polynomial time, so can P_1.

A decision problem $P \in \mathcal{NP}$ is called *$\mathcal{NP}$-complete* if $P' \propto P$ for all $P' \in \mathcal{NP}$, which means that, if an $\mathcal{NP}$-complete problem was solvable in polynomial time, all problems in $\mathcal{NP}$ could be solved in polynomial time, which then would imply that $\mathcal{P} = \mathcal{NP}$. The proof of the existence of an $\mathcal{NP}$-complete problem, which is also known as Cook's Theorem, is one of the most celebrated results and most important milestones (if not the most important milestone) in complexity theory. Finally, an optimization problem is called *$\mathcal{NP}$-hard* if its associated decision problem is $\mathcal{NP}$-complete. A problem is said to be *strongly $\mathcal{NP}$-complete ($\mathcal{NP}$-hard)* if it is $\mathcal{NP}$-complete ($\mathcal{NP}$-hard) when the input is unary encoded instead of binary. If it should be expressed that a problem is $\mathcal{NP}$-complete but *not* strongly $\mathcal{NP}$-complete, we call the problem *weakly $\mathcal{NP}$-complete.*

2.3 Graphs and Networks

Many problems in the field of discrete optimization are motivated from real-world problems, often involving geographic information. Graphs and networks are a highly vivid way of expressing these problems and are used extensively throughout this thesis. For a more detailed overview, we refer to [AMO93; Die97].

Basic concepts A *directed graph* $H = (V, R)$ consists of a nonempty, finite set V of *vertices* or *nodes* and a finite, possibly empty set R of *arcs*. Each arc $r \in R$ has a *start node* denoted by $\alpha(r) \in V$ and an *end node* denoted by $\omega(r) \in V$. Similarly, an *undirected graph* $H = (V, E)$ consists of a nonempty, finite set V of *vertices* or *nodes* and a finite, possibly empty set E of *edges*, where each edge is a pair of nodes. Since most graphs in this thesis are directed, we restrict the provided overview to directed graphs, while the most definitions and concepts carry over similarly to undirected graphs. Further, if the context is clear, we omit the "directed" or "undirected" when referring to a graph. Within this thesis, we usually denote a graph by G if it is the input graph of a problem. When we explicitly want results to hold for a larger class of graphs, the graph is denoted by H.

For the remainder of this section, let $H = (V, R)$ be a directed graph. Whenever the graph is clear from the context, we denote by n its number of nodes and by m its number of arcs (or edges in the undirected case). Two arcs $r_1, r_2 \in R$ are called *parallel* if $\alpha(r_1) = \alpha(r_2)$ and $\omega(r_1) = \omega(r_2)$ and an arc $r \in R$ is called a *self-loop* if $\alpha(r) = \omega(r)$.

Adjacency and Incidence A node $v \in V$ and an arc $r \in R$ are called *incident* if $v \in \{\alpha(r), \omega(r)\}$. Two nodes $u \neq v$ are called *adjacent* if there exists an arc $r \in R$ that is incident to both u and v. Further, two arcs $r_1, r_2 \in R$ are called *incident* if there exists a node to which both arcs are incident. For a node $v \in V$, we denote by $\delta^-_H(v)$ the set of its *incoming arcs*, i.e., $\delta^-_H(v) := \{r \in R \,|\, \omega(r) = v\}$ and by $\delta^+_H(v)$ the set of its *outgoing arcs*, i.e., $\delta^+_H(v) := \{r \in R \,|\, \alpha(r) = v\}$. Further, the set of incident arcs of v are denoted by $\delta_H(v) := \delta^-_H(v) \cup \delta^+_H(v)$. Whenever the graph is clear from the context, we omit the H in the index. For a subset $V' \subseteq V$ of nodes, we define $\delta^-_H(V') := \{r \in R \,|\, \omega(r) \in V' \text{ and } \alpha(r) \notin V'\}$ and analogously $\delta^+_H(V') := \{r \in R \,|\, \alpha(r) \in V' \text{ and } \omega(r) \notin V'\}$.

Paths A common task in graph-related problems is to navigate trough a graph cost-efficiently, which motivates the following definitions. A *(directed) path* $P = (r_1, \ldots, r_k)$ is a finite sequence of arcs such that, for each $i \in \{1, \dots, k-1\}$, it holds that $\omega(r_i) = \alpha(r_{i+1})$, i.e., the end node of each arc is the start node of the next arc in the path. We call $\alpha(r_1)$ the *start node* of the path and $\omega(r_k)$ the *end node* of the path. The *trace* of P is defined as $\operatorname{trace}(P) := (\alpha(r_1), \dots, \alpha(r_k), \omega(r_k))$, i.e., the sequence consisting of the nodes that are visited by the path. An *undirected path* $P = (v_1, r_1, v_2, \dots, v_k, r_k, v_{k+1})$ is a finite sequence with $v_1, \dots, v_{k+1} \in V$ and $r_1, \dots, r_k \in R$ such that, for each $i \in \{1, \dots, k\}$, the arc r_i connects v_i and v_{i+1}, or, intuitively speaking, an undirected path is a path that ignores the direction of the arcs. The sequence $(v_1, \dots, v_{k+1})$ is then called the *trace* of the undirected path. Whenever we speak of a path, we mean a directed path.

In the following, let $P = (r_1, \dots, r_k)$ be a path and $(v_1, \dots, v_{k+1})$ be its trace. For two nodes $s, t \in V$, the path P is called an *s-t-path* if $\alpha(r_1) = s$ and $\omega(r_k) = t$, i.e., if it starts in s and ends in t. The path P is called *elementary* if, for any two nodes v_i and v_j in the trace of P, it holds that $v_i = v_j$ implies that $i, j \in \{1, k+1\}$. If $v_1 = v_{k+1}$, the path P is called a *cycle*.

Let $u, v \in V$ be two nodes. We call u a *parent* of v if there exists an arc from u to v and, reversely, call v a *child* of u in this case. Further, u is called a *predecessor* of v if there exists a path from u to v and, reversely, v is called a *successor* of u in this case. A node without children is called *leaf* and a node without parents is called *root*. A graph, in which two nodes are connected by exactly one undirected path is called a *tree*.
A *binary tree* is defined recursively as follows:

- The graph consisting of a single node is a binary tree.
- For two binary trees T_1 and T_2, the graph consisting of T_1, T_2, and a node v that has an arc to each of the root nodes of T_1 and T_2 is a binary tree.

Note that, by construction, each binary tree has one unique root node. A node of a binary tree that is not a leaf is called an *inner node*.

Graph properties A directed graph without self-loops and parallel arcs is called *simple*. A graph is called *connected* if, for each pair $u, v \in V$ of nodes, there exists a path from u to v and a graph is called *weakly connected* if, for each pair $u, v \in V$ of nodes, there exists an undirected path from u to v. Further, a graph which does not have a cycle is called *acyclic*. A topological sorting of the nodes is a bijection $\sigma : V \to \{1, \dots, n\}$ such that, for each arc $r \in R$, it holds that $\sigma(\alpha(r)) < \sigma(\omega(r))$. It is well known that a graph admits a topological sorting if and only if it is acyclic.

Subgraphs Solving graph-related optimization problems efficiently often requires solving subproblems on specific parts of the graph. This motivates the introduction of so-called subgraphs. A graph $H' = (V', R')$ is a *subgraph* of $H = (V, R)$ if $V' \subseteq V$ and $R' \subseteq R$. For a subset $\bar{V} \subseteq V$ of nodes, the graph $H - \bar{V}$ is defined as the graph obtained from H by removing the nodes in $\bar{V}$ and all their incident arcs. Analogously, for a subset $\bar{R} \subseteq R$ of arcs, the graph $H - \bar{R}$ is defined as the graph obtained from H by removing the arcs in $\bar{R}$. For a subset $V' \subseteq V$, we define the *subgraph of H induced by V'* by the subgraph of H that contains exactly the nodes in V' and the arcs that are not incident to nodes in $V \backslash V'$ and denote it by $H|_{V'}$. Similarly, for a subset $R' \subseteq R$ of arcs, we define the *subgraph of H induced by R'* by the subgraph of H that contains exactly the arcs in R' and their incident nodes and denote it by $H|_{R'}$. A *(weakly) connected component* of H is an inclusionwise-maximal (weakly) connected subgraph of H.

Networks and flows When distributing goods on a graph, it is often a reasonable assumption that the amount of goods that is sent over an arc is limited. A *network* $H = (V, R, u)$ is a directed graph extended by a *capacity function* $u : R \to \mathbb{R}_{\geqslant 0}$. The *capacity* of a set $\bar{R} \subseteq R$ of arcs is defined as $u_H(\bar{R}) := \sum_{r \in \bar{R}} u(r)$. Given a *demand function* $b : V \to \mathbb{R}$ with $\sum_{v \in V} b(v) = 0$, a *flow satisfying the demand* is a function $f : R \to \mathbb{R}_{\geqslant 0}$ that satisfies $b(v) = \sum_{r \in \delta^-(v)} f(r) - \sum_{r \in \delta^+(v)} f(r)$ for each $v \in V$ and $f(r) \leqslant u(r)$ for each $r \in R$. For two nodes $s \neq t$, an *s-t-flow* is a flow that satisfies a demand function b that is of the form $b(v) = 0$ for all $v \in V \backslash \{s, t\}$, $b(s) = -\text{val}$, and $b(t) = \text{val}$ for some val $\in \mathbb{R}_{\geqslant 0}$, which is called the *value* of the s-t-flow. If the nodes s and t are clear from the context, we simply speak of a flow.

A *maximum s-t-flow* is an s-t-flow of maximum value. An *s-t-cut*, or *cut* if s and t are clear from the context, is a partition $V = S \mathbin{\dot\cup} T$ of the nodes of H such that $s \in S$ and $t \in T$ and we say that the arcs in $\delta^+(S)$ are in the cut $C = (S, T)$. For an arc r being in a cut C, we slightly abuse notation and write $r \in C$. The *capacity* of a cut $C = (S, T)$ in H is defined as $\text{cap}^H(C) := \sum_{r \in C} u(r)$, and a *minimum cut* in H is a cut of minimum capacity in H. The celebrated max-flow min-cut theorem states that the value of a maximum flow is equal to the capacity of a minimum cut in H.

2.4 Multicriteria Optimization

In some optimization problems, there are multiple, possibly conflicting, criteria contributing to the quality of a solution. Take the example of buying a new car. On the one hand, you want the car to be cheap, but on the other hand, you also want your car to be fuel-efficient (or power-efficient if it is electric). We briefly introduce the concept of multicriteria optimization and refer to [Ehr05] for a more fundamental description.

Pareto orders For two vectors $y, y' \in \mathbb{R}^p$ for some $p \geqslant 2$, we define the *Pareto order* as follows:

$$y \leqslant_{\text{P}} y' :\Longleftrightarrow y_i \leqslant y'_i \text{ for all } i \in \{1, \ldots, p\} \text{ and } y_j < y'_j \text{ for some } j \in \{1, \ldots, p\}$$

Given a set $\mathcal{Y} \subseteq \mathbb{R}^p$, a vector $y \in \mathcal{Y}$ is called *non-dominated with respect to maximization (minimization)* if there does not exist another vector $y' \in \mathcal{Y}$ with $y \leqslant_{\text{P}} y'$ ($y' \leqslant_{\text{P}} y$).

Multi-objective problems An *instance* of a multi-objective optimization problem consists of a set $\mathcal{X}$ of feasible solutions, an objective function $f : \mathcal{X} \to \mathbb{R}^p$, where we refer to the i-th component of f by f^i, and an *optimization sense* max or min. A *multi-objective optimization problem* is given by the set of its instances. The image $\mathcal{Y} := f(\mathcal{X})$ of f is called the *image set.* The aim in an instance of a multi-objective optimization problem is to find a subset $X \subseteq \mathcal{X}$ of solutions, called an *efficient set* such that its image $Y := f(X)$ under f is the set of all non-dominated points in $\mathcal{Y}$ with respect to the optimization sense. A solution x, for which $y = f(x)$ is non-dominated in $\mathcal{Y}$ is called *efficient.* To distinguish between the multiple criteria and a one-dimensional objective function as introduced in Section 2.1, we refer to problems of the latter type as *single-objective problems.*

2.5 Approximation Algorithms

As we have seen in Section 2.2, there exist optimization problems that, unless $\mathcal{P} = \mathcal{NP}$, cannot be solved to optimality in polynomial time. Some of those problems, however, admit polynomial-time algorithms that return a solution whose objective value is guaranteed to deviate no more than a certain multiplicative factor from the objective value of an optimal solution. Such algorithms are called approximation algorithms. For a detailed overview on approximation algorithms, the reader is referred to [Vaz01; WS11] for the single-objective case and to [HRT21] for the multi-objective case. Whenever we refer to approximation algorithms throughout this thesis, we implicitly assume that all feasible solutions have a non-negative objective value.

Single-objective approximations Let P be a single-objective maximization (minimization) problem and let $\alpha \geqslant 1$. An algorithm ALG for P that computes a feasible solution with objective value at least $1/\alpha$ (at most α) times the optimal objective value in polynomial time for every instance is called an *α-approximation algorithm.* The value α is then called the *approximation ratio* of the algorithm.

A *polynomial-time approximation scheme* (PTAS) is a family $(\mathrm{ALG}_\varepsilon)_{\varepsilon>0}$ of algorithms such that, for each $\varepsilon > 0$, the algorithm ALG_ε is a $(1 + \varepsilon)$-approximation algorithm. A PTAS $(\mathrm{ALG}_\varepsilon)_{\varepsilon>0}$ is called a *fully polynomial-time approximation scheme* (FPTAS) if the running time of ALG_ε is additionally polynomial in $1/\varepsilon$ and the encoding length of the problem instance for all $\varepsilon > 0$.

Multi-objective approximations A similar concept can be introduced for multi-objective problems. It is worth mentioning that the number of non-dominated points may be super-polynomially large for most discrete multi-objective optimization problems. Hence, even if $\mathcal{P} = \mathcal{NP}$, one cannot compute an efficient set for these problems in polynomial time, which is another motivation to introduce multi-objective approximations. To this end, let $\alpha \geqslant 1$ and let P be a multi-objective problem with feasible set $\mathcal{X}$, objective function $f : \mathcal{X} \to \mathbb{R}^p$, and image set $\mathcal{Y} = f(\mathcal{X})$. A feasible solution $x \in \mathcal{X}$ *α-approximates* another feasible solution $x' \in \mathcal{X}$ (or, equivalently, the feasible point $y = f(x)$ *α-approximates* the feasible point $y' = f(x')$) if

$$f^i(x) \geqslant 1/\alpha \cdot f^i(x') \text{ for all } i \in \{1, \dots, p\} \text{ if the optimization sense is maximize, or}$$
$$f^i(x) \leqslant \alpha \cdot f^i(x') \text{ for all } i \in \{1, \dots, p\} \text{ if the optimization sense is minimize.}$$

A set $X \subseteq \mathcal{X}$ of feasible solutions is called an *α-approximate Pareto set* if, for every feasible solution $x' \in \mathcal{X}$, there exists a solution $x \in X$ that α-approximates x'.

An *α-approximation algorithm* for P is an algorithm ALG that computes an α-approximate Pareto set in polynomial time for every instance.

Moreover, a *multi-objective polynomial-time approximation scheme* (MPTAS) for the problem P is a family $(\mathrm{ALG}_\varepsilon)_{\varepsilon>0}$ of algorithms such that, for each $\varepsilon > 0$, the algorithm ALG_ε is a $(1 + \varepsilon)$-approximation algorithm. An MPTAS $(\mathrm{ALG}_\varepsilon)_{\varepsilon>0}$ for P is called a *multi-objective fully polynomial-time approximation scheme* (MFPTAS) if the running time of ALG_ε is additionally polynomial in $1/\varepsilon$ and the encoding length of the problem instance for all $\varepsilon > 0$.

3 A Mixed-Integer Programming Approach to Municipal Flood Mitigation

Abstract Adapting to the consequences of climate change is one of the central challenges faced by humanity in the next decades. One of these consequences are intense heavy rain events, which can cause severe damage to buildings due to flooding. In this chapter, we present the first use of optimization techniques that scales well enough to be applicable for supporting decision making in planning precautionary measures for realistic pluvial flash flood scenarios. Our mixed-integer programming model has been implemented as an innovative decision support tool in the form of a web application, which has already been used by more than 30 engineering offices, municipalities, universities, and other institutions. The model aims to minimize the damage caused in the case of a heavy rain event by taking best-possible actions subject to a limited budget and constraints on the cooperation of residents. We further present an efficient, graph-based representation and preprocessing of the surface terrain, a combinatorial algorithm for computing an initial solution of the mixed-integer program, and computational results obtained on real-word data from different municipalities.

3.1 Introduction

Scientists agree that heavy rain events will increase both in their intensity and their frequency within the next years [Ble+18; IPC21; RS17]. Adapting these rain events or, more generally, to the consequences of climate change is assuredly one of the central challenges faced by humanity in the next decades.

The flash flood from 14 to 15 July 2021 in Germany, Luxembourg, and Belgium at the latest has caused a special public interest in adapting to such events. This event claimed more than 180 lives [FS21] and caused tremendous damage, which has been been estimated at a total of 32 billion euros [Moh+23].

Typically, flood mitigation concepts are created based on simulations rather than using optimization methods [Ger16]. Prioritization of actions is then often conducted by a simple point scheme [Sie18]. This method, like any other purely simulation-based method, lacks the consideration of site-specific interplay of actions, which motivates using optimization methods in the context of flood mitigation.

3.1.1 Previous Work

Although there is clear evidence of the efficacy of precautionary measures [Kre+05], the literature on the use of optimization techniques in order to design flood mitigation concepts is surprisingly limited, which is also pointed out in [Tas21; WKG14]. The closest related study to this work is [Tas21], where the "Optimal Flood Mitigation Problem", in which the aim is to optimize the positioning of a single type of precautionary measure (embankments) to protect critical assets in the case of a flood scenario, is introduced. Further, two time-indexed mixed-integer programming formulations that over- and underestimate the water flows during a flooding scenario are presented. Here, the over- and underestimation is caused by the linearization of nonlinear constraints. Due to the time-indexed formulation, however, the MIPs do not tractably scale to realistic scenarios (as noted in the abstract of [Tas21]).

The problem of designing mitigation concepts for coastal floods in the Netherlands is an impressive example of the potential of using optimization techniques in flood mitigation. A mixed-integer programming formulation for a cost-efficient design of dike heights is presented in [Bre+12; ZVH18]. Further, a greedy search algorithm to compute a combination of reinforcement measures for dike segments, which is 42% cheaper than the combination obtained from the common approach, is implemented in [Kle+21].

Moreover, a genetic algorithm is used to compute efficient mitigation concepts for fluvial (river-caused) flash floods on the Thames Estuary (London, England) in a multi-objective setting [WKG14]. Apart from the measures themselves, they also compute a threshold value for the timing to make an intervention given the uncertainty of the development of climate change and its impact on fluvial flash floods.

Another approach to the design of flood mitigation concepts can be found in [Hua+18], where a simulated annealing algorithm is used to determine an allocation of low-impact actions such as porous pavements, green roofs, etc. to districts in a megacity. Moreover, a particle swarm optimization algorithm is used in [Ngo+16] to determine an optimal pumping schedule and optimal weir crest heights for detention reservoirs to minimize downstream flood damage.

An often neglected but crucial factor in creating successful flood mitigation concepts is taking the cooperation of residents into account since, in many cases, the most efficient precautionary measures are located on private properties. Indeed, the potential of incentives in flood prevention has already been established as promising [JKS08; MHJ18; PBA14]. In practice, however, plans are often made before involving critical private actors. A holistic review on using market-based instruments for flood risk management is provided in [Fil14].

Beside these approaches for flood mitigation, a wide variety of optimization techniques are used in post-disaster flood management. The design of evacuation plans including shelter location planning and helicopter assignment in a multi-objective robust setting is investigated in [KP21] and real-time operation procedures that specify reservoir releases during a flood are examined in [CM15; WH08]. For a more extensive review of optimization and machine learning approaches in post-disaster flood management, we refer to [Mun+21].

Other applications of optimization techniques in water management involve the design of sewage water systems [Sch+14], a real-time release schedule for reservoirs during a flood [WH08], and the geometrical design of retention basins [Nem+22].

3.1.2 The Project AKUT

The work presented in this chapter has been performed within the project AKUT – an acronym for the German translation of “Incentive Systems for Municipal Flood Prevention” – which has been funded by the German Federal Ministry for the Environment, Nature Conservation, and Nuclear Safety from January 2019 to March 2021. Within the project, a mixed-integer programming approach has been developed to find an optimal combination of precautionary measures to be taken such that the resulting damage on buildings is minimized while respecting a given budget and constraints on the cooperation of the residents. The resulting MIP has been implemented in a web application (also referred to as AKUT) using the Flask framework and Python 3.8. The application is available for municipalities free of charge (so far only in German language).

The project team included a municipality providing us with real-world data and the engineering office “igr GmbH” validating our results by comparing them to results of state-of-the-art simulations. Further, the Professorship of Water Resource Management and Sanitary Environmental Engineering at Mainz University of Applied Sciences formulated and developed the engineering methodology for the model while we formulated the mathematical model and implemented the web application.

We present a novel mixed-integer programming approach for computing optimized flood mitigation concepts that minimize the damage to buildings due to pluvial flash floods. To the best of our knowledge, this approach marks the first usage of optimization techniques in the context of planning precautionary measures for pluvial flood mitigation that scales well enough to be applied to real-world instances. Our model allows for different types of precautionary measures (basins, ditches, and embankments) that lead to elevations or depressions of the terrain surface. Moreover, the model takes constraints on the cooperation of the residents into account. One of the central challenges to make this approach work for realistic scenarios is modeling the terrain surface efficiently while still maintaining a realistic representation. We tackle this challenge via an efficient graph-based approach together with suitable preprocessing methods. Moreover, we present a combinatorial algorithm that is able to quickly compute an initial feasible solution of the presented MIP.

Our approach is implemented as an innovative decision support tool in the form of a web application, which has already been used in practice by more than 30 engineering offices, municipalities, universities, and other institutions from all over Germany. We compare the results obtained from our model on real-world instances from different municipalities to results obtained from established simulation software, and investigate the main drivers for the running time and the quality of the obtained solutions. The novelty of our approach in comparison to a selection of the previously presented existing literature is summarized in Table 3.1.

Reference	pre-/post-disaster	optimization	pluvial	scalable	incentivation
[Tas21]	pre	✓	✓		
[WKG14]	pre	✓		✓	
[Bre+12; ZVH18]	pre	✓		✓	
[CM15; WH08]	post	✓		✓	
[KP21]	post	✓	✓	✓	
[PBA14]	pre		✓		✓
This work	pre	✓	✓	✓	✓

Table 3.1.: Comparison of our work to existing literature. A tick in the column "optimization" indicates that optimization algorithms are used and a tick in the column "pluvial" represents that the paper considers a pluvial flood scenario (as opposed to a fluvial or coastal flood scenario). A tick in the column "scalable" indicates that the developed method scales well enough to be applied to realistic scenarios. Finally, a tick in the column "incentivation" means that incentives or cooperation of residents is considered.

3.2 Problem Description and Input Data

In this section, we define the underlying problem and describe the input data on which our approach is based.

In short, given a set of possible locations for retention basins (simply called basins in the following), ditches, and embankments, the goal in the problem is to determine a subset of these precautionary measures, usually called *actions* in the following, to take such that the resulting damage to buildings is minimized while respecting a given budget and constraints on the cooperation of residents.

The terrain surface is given as a digital terrain model (DTM), which is an established standard in engineering [Ger16]. A DTM contains 2D coordinates in UTM format on a given grid together with their corresponding geodesic height, which is similar to the elevation above sea level. In our case, a grid size of one meter is used.
Each of the data points in the DTM then determines the geodesic height of the one by one meter square centered at the 2D coordinate. This square is called the *shape* of a coordinate, and two 2D coordinates are called *adjacent* if their distance is one meter, i.e., if one coordinate is one meter to the north, south, west, or east of the other. These data are available for all German municipalities and, hence, suitable for applying our model in practice.

To estimate the damage that occurs due to flooding in the case of a rain event, information about the buildings' locations is required. To this end, the *shape* of a building is defined as the polygon derived from its outline. The outlines of the buildings are obtained from ALKIS[1], which is a digital land information system. Similar to the data for the DTM, these data are available to all German municipalities.

[1]German acronym for "official real estate cadastre information system"

In general, we say that two shapes *intersect* if their geometric intersection has a strictly positive area. To link the positions of the buildings to the DTM coordinates, we say that a building is *on* a coordinate, if the shape of the building and the shape of the coordinate intersect. Conversely, we say that a coordinate *intersects* with a building in this case.

The definition of damage caused to buildings is based on the advisory leaflet DWA-M 119 [Ger16] published by the German Association for Water, Wastewater and Waste (DWA) in 2016. The DWA is a politically and economically independent organization that supports safe and sustainable water management and prepares the DWA set of rules, which includes a large number of standards and advisory leaflets. Within the advisory leaflet DWA-M 119, they identify two main factors for the damage caused to a building in case of a flash flood.

The first main factor is the maximum water level that occurs at a building. Here, the *maximum water level* at a building is defined as the maximum over all water levels at the coordinates intersecting with the building. The *hazard class*, which represents the maximum water level at a building, is a categorical measure attaining the values zero to four. It is derived from the maximum water level at a building by the following rules:[2]

- **Zero:** The maximum water level at a building is 0cm, i.e., none of the coordinates intersecting with the building has a strictly positive water level.
- **One:** The maximum water level at the building is strictly larger than 0cm and less than or equal to 10cm.
- **Two:** The maximum water level at the building is strictly larger than 10cm and less than or equal to 30cm.
- **Three:** The maximum water level at the building is strictly larger than 30cm and less than or equal to 50cm.
- **Four:** The maximum water level at the building is strictly larger than 50cm.

The second main factor describes the (quite intuitive) fact that not every building suffers an equal amount of damage at a given water level. As an example, it is by far less severe if a garage is affected by the rain event compared to the case where a hospital is affected. To take this into account, the damage at a building does not only depend on the water level at the building (represented by its hazard class), but also on its *damage class*. The damage class is a categorical measure of the damage occurring at a building and can attain the values one to four, where one corresponds to the lowest damage class (the garage in our example), i.e., the least amount of damage, and four corresponds to the highest damage class (the hospital in our example). The data from ALKIS, aside from just the shape of the building, provide additional information about the buildings like their usage, which allows to preset the damage class for some of the buildings automatically. For the remaining buildings, the damage class has to be specified manually.

The combination of the hazard class and the damage class yields the *need for protection* of a building, which is rated using a point system with a scale from zero to seven. For buildings

[2]See advisory leaflet DWA-M 119 [Ger16].

with hazard class zero, i.e., none of the coordinates intersecting with the building have a strictly positive water level, the need for protection is also zero. Every other building has a strictly positive need for protection, which is obtained by taking the sum of the hazard- and danger class minus one. The objective of the problem is to minimize the sum of all buildings' needs for protection.

In order to protect the buildings, a set of potential basins, ditches, and embankments is given, which together make up the possible *actions*. Each possible action is given by the polygon of its location, its construction costs, and its depth (in the case of a basin or ditch) or height (in the case of an embankment). The polygon of an action's location is called its *shape*.

Similar to the buildings, we say that an action is *on* a coordinate if its shape intersects with the shape of the coordinate. In this case, we also say that the coordinate *intersects* with the action. If an action is taken, i.e., a basin, ditch, or embankment is built, the geodesic height of all coordinates intersecting with the action is decreased by the action's depth or increased by its height. The change in geodesic height affects the flow of the water on the terrain surface and, hence, can protect buildings. The overall cost for taking actions is bounded from above by a given budget.

Taking an action sometimes requires the consent of the owners of the properties on which the action is located. The owners of the properties are called *actors* in the following. The outlines of the properties are also obtained from ALKIS. As before, the *shape* of a property is defined as the polygon derived from its outline. An action is *on* a property if their shapes intersect. Convincing actors to cooperate might be more or less hard. To guarantee that the recommended combination of actions can realistically be implemented, the number of hard-to-convince actors on whose properties actions are to be taken is bounded from above. To this end, an extended traffic light rating system with the following characterizations is used:

- **Green:** The actor is willing to cooperate.
- **Yellow:** The actor needs minor incentives to cooperate.
- **Red:** The actor needs major incentives to cooperate.
- **Black:** The actor does not cooperate at all.

For simplicity, we also refer to *green, yellow, red*, and *black properties* in the following. The willingness to cooperate has to be assigned manually by the user for each property.

3.3 Mathematical Modeling

We now present a graph-based model for the problem described in Section 3.2 as well as an approach for reducing the size of the underlying graph. Afterwards, we derive our mixed-integer programming formulation that is used for solving the graph-based model and we describe valid inequalities and presolve techniques that are used to improve performance.

3.3.1 Graph-Based Model

In this section, we derive a graph-based model for the problem introduced in Section 3.2. To this end, we start by presenting how a graph can be constructed from the input data and proceed by presenting an efficient algorithm to compute the water levels for a given combination of selected actions. Finally, we show how the graph size can be reduced while maintaining a realistic representation of the real-world instance.

3.3.1.1 Construction of the Graph

In this section, we construct the directed graph $G_{\text{or}} = (V_{\text{or}}, R_{\text{or}})$, which we call the *original graph*, from the DTM.

For the construction of $G_{\text{or}} = (V_{\text{or}}, R_{\text{or}})$, recall that the DTM contains data about the geodesic height for coordinates on a one meter grid. The set V_{or} of nodes is constructed by associating one node with each of these coordinates, i.e., there is a one to one correspondence between the coordinates in the DTM and the nodes in the graph. To keep track of this correspondence, each node gets the coordinate as an additional attribute.

The *geodesic height* of a node is defined as the geodesic height of its corresponding coordinate and is stored as an attribute of the node in the graph. We then index the nodes in $V_{\text{or}} = \{v_1, \dots, v_n\}$ in non-decreasing order of geodesic heights, where ties are broken arbitrarily. Further, we define the *shape* of a node $v \in V_{\text{or}}$ as the shape of its corresponding coordinate, i.e., in this case, the one meter square with its center at the corresponding coordinate. The definitions of whether a building, action, or property intersects with (the shape of) a node are analogous to the ones for coordinates provided in the previous section. Finally, each node $v \in V_{\text{or}}$ is assigned an *area*, which we denote by $\text{area}(v)$. In the case of the original graph, the area is one square meter for each node. This changes for the graphs that are constructed in Section 3.3.1.3, however.

For any two nodes whose corresponding coordinates are adjacent on the grid, there is an arc in R_{or} between the nodes, which is oriented from the node with the higher index to the node with the lower index. This means that arcs are directed from the node with larger geodesic height (the *higher* node) to the node with lower geodesic height (the *lower* node) whenever the two nodes do not have the same geodesic height. Note that, if all nodes in V_{or} have pairwise distinct geodesic heights, this makes the original graph $G_{\text{or}} = (V_{\text{or}}, R_{\text{or}})$ acyclic since the geodesic heights induce a topological sorting (both in the mathematical and literal sense) in this case. An example of the original graph is provided in Figure 3.1.

To model the runoff behavior of the precipitation water, we compute flows on the graph, which are determined by the nodes' geodesic heights. To this end, for an arc $r \in R_{\text{or}}$, we define its *slope* as the absolute difference of its incident nodes' geodesic heights and denote it by $\text{slope}(r)$. When distributing the outflow of a node $v \in V_{\text{or}}$ among its downhill arcs, we want to ensure that a higher slope causes more water flow on an arc. This is modeled by the *ratios* of the arcs, which are introduced next and are based on the concept of *processing* *networks* [HKT17; Koe83], in which flow is distributed proportionally among the outgoing arcs of a node according to fixed ratios.

To compute the ratio of an arc $r \in R_{\text{or}}$, which we denote by $\text{ratio}(r)$, we distinguish two cases. If the sum of the slopes of all outgoing arcs of the node $\alpha(r)$ is nonzero, we define the ratio of r by $\text{ratio}(r) := \text{slope}(r)/\sum_{\hat{r}\in\delta^+(\alpha(r))} \text{slope}(\hat{r})$. If the sum in the denominator is zero, i.e., if all successors of $\alpha(r)$ have the same geodesic height as $\alpha(r)$, the ratio of r is defined as one divided by the number of successors, i.e., as $\text{ratio}(r) := 1/|\delta^+(\alpha(r))|$.

In some situations, however, actions that are taken lead to water flowing in the opposite direction of an arc $r \in R_{\text{or}}$, which means that the original graph does not suffice for our model. A simple example for such a situation is illustrated in Figure 3.2. To this end, for an arc $r \in R_{\text{or}}$, we denote the inverse arc by $\overleftarrow{r}$. The *extended original graph* $G^{\text{ex}}_{\text{or}} = (V_{\text{or}}, R^{\text{ex}}_{\text{or}})$ is then constructed by adding the inverse arc $\overleftarrow{r}$ for each $r \in R_{\text{or}}$ to the original graph and setting the ratio of the arc $\overleftarrow{r}$ to the ratio of r. Note that this does not change the node set V_{or}.

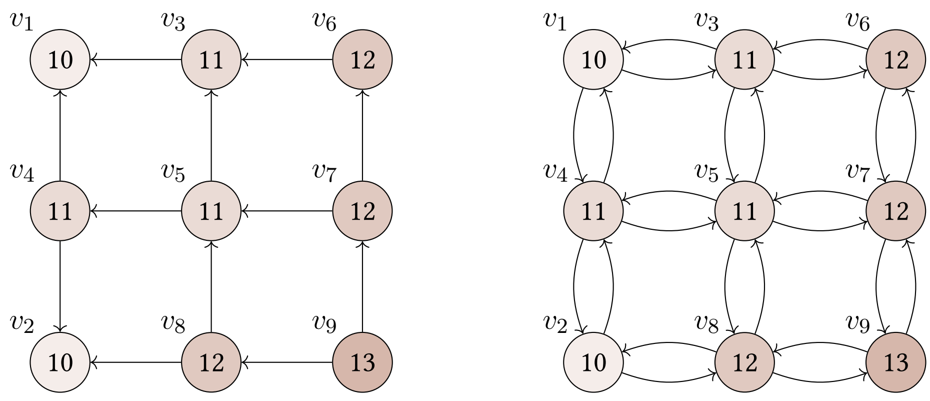


Figure 3.1.: An example of the original graph $G_{\text{or}} = (V_{\text{or}}, R_{\text{or}})$ on the left, and an example of the extended original graph $G^{\text{ex}}_{\text{or}} = (V_{\text{or}}, R^{\text{ex}}_{\text{or}})$ on the right. The number in each node corresponds to its geodesic height, also indicated by the node's color The nodes are indexed in non-decreasing order of geodesic height, where ties are broken arbitrarily as, e.g., for v_4 and v_5. The arcs in the original graph are directed such that they start at the node with the higher index.

3.3.1.2 Description of the Graph-Based Model

The goal in our problem is to provide best-possible protection for the buildings by taking a combination of actions respecting a given budget and the cooperation of the actors. In this section, we formulate the corresponding optimization problem formally by using the extended original graph introduced in the previous section.

The input of our graph-based model consists of:

- The **original graph** $G_{\text{or}} = (V_{\text{or}}, R_{\text{or}})$ and the **extended original graph** $G^{\text{ex}}_{\text{or}} = (V_{\text{or}}, R^{\text{ex}}_{\text{or}})$
- The set B of **buildings**, each of which is given by its shape and its damage class

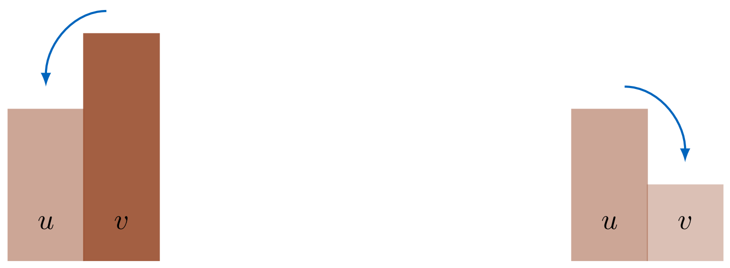


Figure 3.2.: The instance consists of two nodes u and v, where v is the higher of the two nodes. This means that water flows from v to u, which is illustrated on the left-hand side. If a basin with depth strictly larger than the absolute difference of the nodes' geodesic heights is built on v, the resulting geodesic height of v after building the basin is less than the geodesic height of u. Therefore, the water flows in the opposite direction after building the basin, which is illustrated on the right-hand side.

- The set $\mathcal{A}$ of possible **actions**, each of which is given by its shape, its construction costs, and its depth/height
- The set P of **properties**, each of which is given by its shape and the willingness to cooperate of the corresponding actor
- A **budget**, also denoted by budget, which represents an upper bound on the total cost for taking actions
- The **maximum combined number of yellow and red properties**, which is denoted by maxAllowedYellow + maxAllowedRed, on which actions can be taken
- The **maximum number of red properties** maxAllowedRed on which actions can be taken
- The **rain** per square meter (sqm) denoted by rain

A feasible solution is a set of actions whose total cost does not exceed the given budget and where neither the combined number of yellow and red properties on which actions are taken nor the number of red properties on which actions are taken exceeds the allowed maximum. The objective is to minimize the sum of all buildings' needs for protection, which is computed from a given feasible solution as described in the following.

The decision on which actions are to be taken changes the geodesic heights of the nodes intersecting with these actions, which may in turn change the flows in the graph. The change of the geodesic height is straightforward if there is at most one action taken on a node. However, if there are several actions with different depths/heights taken on one node, like for example a ditch leading into a deeper basin, we need a more sophisticated rule, which is given as follows:

(GH1) If at least one action decreasing the geodesic height (i.e., a basin or a ditch) is built on a node $v \in V_{\text{or}}$, then the geodesic height of v is set to the node's original geodesic height minus the maximum depth of any of the basins or ditches built on v.

(GH2) If no actions decreasing the geodesic height are built on a node $v \in V_{\text{or}}$ (i.e., neither basins nor ditches are built on v) but an embankment is, then the geodesic height of v is set to the node's original geodesic height plus the maximum height of any of the embankments built on v.

Once the selection of the taken actions has been made, the resulting geodesic heights (after taking the actions) determine the flows between the nodes, which then allows us to compute the water levels. Before we describe how the flows are computed, we describe the connection between the flows and the water levels. To this end, we define the *excess* of a node $v \in V_{\text{or}}$ as the amount of water accumulating at v, i.e., as the initial water from the rainfall plus the node's inflow minus its outflow. The *water level* at v is defined as the excess of v divided by the node's area.

We next describe how the water levels are computed. An efficient implementation of a combinatorial algorithm for this computation is provided as Algorithm 3, which uses Algorithms 1 and 2 as subroutines for computing the flows in the graph and joining nodes, respectively. To this end, let $G = (V, R)$ be the input graph of the problem, i.e., the graph G_{or}. It is important to note that the presented algorithm also works for the graphs that are constructed later in Section 3.3.1.3. For a subset $D \subseteq \mathcal{A}$ of actions, we define $G^D = (V^D, R^D)$ as the graph that is obtained from G when adjusting the geodesic heights as described in (GH1) and (GH2) and changing arc directions where necessary. This graph then represents the input to Algorithm 3 when computing the water levels in the scenario where exactly the actions in D are taken. Throughout the computation, we keep track of both the water levels and the excesses of all nodes in V. Initially, a copy $\tilde{G}_1 = (\tilde{V}_1, \tilde{R}_1)$ of the input graph is saved. This graph is modified in each iteration of the algorithm and we denote the graph at the beginning of the t-th iteration of the algorithm by $\tilde{G}_t = (\tilde{V}_t, \tilde{R}_t)$.

To compute the flows in each iteration, Algorithm 1 is called as a subroutine, which gets the graph $\tilde{G}_t = (\tilde{V}_t, \tilde{R}_t)$ in the current iteration as an input. Initially, each node $v \in \tilde{V}_t$ receives its initial *water to be distributed* $\text{wtbd}(v)$ from the rainfall, which is computed by multiplying the given rain per square meter (sqm) with the node's area. This water may then flow over the node's outgoing arcs. To compute the flows, the water to be distributed is distributed proportionally to the ratios of the outgoing arcs if v is not a leaf. This is done by starting at the highest root node in the graph and dispensing all its water to its children according to the ratios of the corresponding arcs. The water to be distributed at its children is updated accordingly. Then, this root node is removed from the graph and the process is repeated with the next highest root node in the graph until all nodes have been processed.

With the flows that are computed using Algorithm 1 in the first iteration or its later-described more efficient version using the update method in the following iterations, we compute in each iteration t of Algorithm 3 for each leaf the *proportion* p_t of the total rain event that is needed to fill up the water level at the leaf to the geodesic height of its parent with lowest geodesic height breaking ties by the indexing of the nodes, which we call the *lowest parent*. The leaf for which the least such proportion is needed is called the *first flooded leaf*.[3] For a node $v \in V$, we denote its lowest parent in $\tilde{G}_t$ by $\text{lp}_{\tilde{G}_t}(v)$. If the graph is clear from the context, we omit the graph in the index.

[3]In case that several leaves have the least proportion, any of them can be designated as the first flooded leaf.

Algorithm 1: COMPUTE-FLOWS

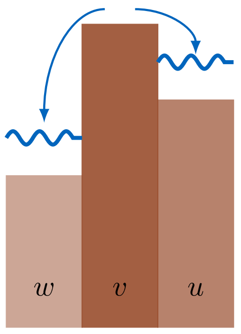
```
Procedure computeFlows($\tilde{G}_t = (\tilde{V}_t, \tilde{R}_t)$)
    Initialize $f(r) = 0$ for all $r \in \tilde{R}_t$
    Initialize wtbd($v$) = rain $\cdot$ area($v$) for all $v \in \tilde{V}_t$
    Save a copy $G' = (V', R')$ of the input graph $\tilde{G}_t = (\tilde{V}_t, \tilde{R}_t)$
    while $V' \neq \emptyset$ do
        Choose $v \in V'$ as a root node with largest geodesic height among all roots in $G'$
        if $v$ is not a leaf in $G'$ then
            Distribute the whole water to be distributed of $v$ among its outgoing arcs
              proportionally to their ratios and save the flow on $r$ in $f(r)$ for each
              $r \in \delta^+_{G'}(v)$
            for $r \in \delta^+_{G'}(v)$ do
                wtbd($\omega(r)$) = wtbd($\omega(r)$) + $f(r)$
        Remove $v$ from $V'$
    return $f = (f(r))_{r \in \tilde{R}_t}$
```

Algorithm 2: JOIN-NODES

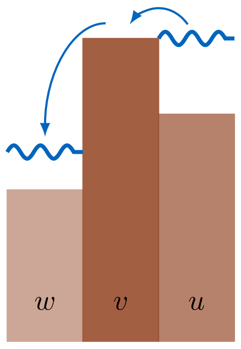
```
Procedure joinNodes($u, v$)
    Save a copy $\tilde{G}_{t+1} = (\tilde{V}_{t+1}, \tilde{R}_{t+1})$ of $\tilde{G}_t = (\tilde{V}_t, \tilde{R}_t)$
    Remove all arcs that have starting node $v$ and terminal node $u$ from $\tilde{R}_{t+1}$
    for $r \in \delta^-_{\tilde{G}_{t+1}}(u)$ with $\alpha(r) \neq v$ do
        Set $\omega(r) = v$ in $\tilde{G}_{t+1}$
    repres$_{t+1}(v)$ = repres$_t(v) \cup$ repres$_t(u)$
    area$_{t+1}(v)$ = area$_t(v)$ + area$_t(u)$
    Remove $u$ from $\tilde{V}_{t+1}$
    for $v' \in \tilde{V}_{t+1} \setminus \{v\}$ do
        repres$_{t+1}(v')$ = repres$_t(v')$
        area$_{t+1}(v')$ = area$_t(v')$
```

Algorithm 3: COMPUTE-WATER-LEVELS

```
1  Procedure computeWaterLevels($G = (V, R)$)
2      Initialize $t = 1$ and $\mathrm{sp}_0 = 0$
3      Save a copy $\tilde{G}_1 = (\tilde{V}_1, \tilde{R}_1)$ of the input graph
4      for $v \in V$ do
5          $\mathrm{repres}_1(v) = \{v\}$
6          $\mathrm{area}_1(v) = \mathrm{area}(v)$
7          $\mathrm{excess}_0(v) = 0$
8      while $\mathrm{sp}_t < 1$ do
9          For each leaf $u \in \tilde{V}_t$, compute its incoming water over the whole rain event
             given that the graph remains unchanged as
                 $$\mathrm{iwr}_t(u) = \mathrm{area}_t(u) \cdot \mathrm{rain} + \sum_{r \in \delta^-_{\tilde{G}_t}(u)} f_t(r),$$
             where the flows $(f_t(r))_{r \in \tilde{R}_t}$ are the flows computed by Algorithm 1 in the first
             iteration $t = 1$ or its modified version using the update method in all later
             iterations $t > 1$
10         For each leaf $u \in \tilde{V}_t$, compute the proportion $p_t(u)$ of the rain event that is
             needed to fill up the water level at the leaf to the absolute difference of its own
             geodesic height and the geodesic height of its lowest parent $\mathrm{lp}(u)$, i.e., such
             that $\mathrm{excess}_{t-1}(u) + \mathrm{iwr}_t(u) \cdot p_t(u) = \big(\mathrm{gh}(\mathrm{lp}_{\tilde{G}_t}(u)) - \mathrm{gh}(u)\big) \cdot \mathrm{area}_t(u)$
11         Denote by $\hat{u}$ a leaf whose corresponding proportion $p_t := p_t(\hat{u})$ is the lowest
             among the computed proportions where ties are broken arbitrarily, and by $\hat{v}$
             its lowest parent
12         if $\mathrm{sp}_{t-1} + p_t \geqslant 1$ then
13             $p_t = 1 - \mathrm{sp}_{t-1}$ and $\mathrm{sp}_t = 1$
14         else
15             $\mathrm{sp}_t = \mathrm{sp}_{t-1} + p_t$
16         For each leaf node $u \in \tilde{V}$, set $\mathrm{excess}_t(u) = \mathrm{excess}_{t-1}(u) + p_t \cdot \mathrm{iwr}(u)$
17         if $\mathrm{sp}_t \neq 1$ then
18             Call joinNodes($\hat{u}, \hat{v}$)
19             $t = t + 1$
20     for $\tilde{v} \in \tilde{V}_t$ do
21         for $v \in \mathrm{repres}_t(\tilde{v})$ do
22             $\mathrm{wl}(v) = {}^{\mathrm{excess}_t(\tilde{v})}/_{\mathrm{area}_t(\tilde{v})} + \mathrm{gh}(\tilde{v}) - \mathrm{gh}(v)$
23     return $\mathrm{wl} = (\mathrm{wl}(v))_{v \in V}$
```



(a) The water at node v is distributed to both its children u and w. Neither of the childrens' water levels is equal to absolute difference of their geodesic height and the geodesic height of v, so the water at v flows from v to the nodes u and w.



(b) The water level at node u is equal to the absolute difference of the geodesic heights of u and v, which means that the nodes u and v are joined and afterwards represented by v. As the water level at w is still less than the absolute difference of the geodesic heights of w and v, all water at v now flows to w.

Figure 3.3.: An illustration of the flows and joining two nodes during Algorithm 3.

The first flooded leaf is then joined into its lowest parent by using Algorithm 2. The excesses until then are saved, and, from there on, the water is increased until either (1) the water level at the next leaf reaches the absolute difference of its own geodesic height and the geodesic height of its lowest parent in the graph $\tilde{G}_t$ of the current iteration of the algorithm, or (2) the sum of the proportions until iteration t, which we denote by sp_t,[4] equals or exceeds one, i.e., we have simulated the whole rain event.

Throughout this process, for each node v, we store the nodes in the input graph that are represented by v in the current iteration t in a set $\mathrm{repres}_t(v)$, which is updated within Algorithm 2. The behavior of the flows as well as joining two nodes during the algorithm is illustrated in Figure 3.3.

It is worth noting that, when u is removed from $\tilde{V}_{t+1}$ in line 8 of Algorithm 2, u has no incident arcs. It has no outgoing arcs because the algorithm is only called for u being a leaf in $\tilde{G}_t$, and the incoming arcs are removed from the graph in line 3 or redirected in the for loop starting in line 4. It is further worth noting that subsequent calls of this routine, as it is done in Algorithm 3, can lead to parallel arcs in the graph.

After the whole rain event has been simulated in Algorithm 3, we recompute the water levels of all nodes in the input graph using the sets $\mathrm{repres}_t(v)$ for $v \in V$, which is done by setting the water level at each node v that has been removed from the graph during the algorithm to ${}^{\mathrm{excess}_T(\tilde{v})}/_{\mathrm{area}_T(\tilde{v})} + \mathrm{gh}(\tilde{v}) - \mathrm{gh}(v)$, where $\tilde{v}$ is the unique node such that $v \in \mathrm{repres}_T(\tilde{v})$ with T denoting the number of iterations of the algorithm.[5] The maximum water level at each of the buildings and, hence, its resulting hazard class, obtained as described in Section 3.2, follow immediately. The combination of the buildings' hazard and damage classes yield the corresponding needs for protections whose sum is to be minimized.

[4]**sp:** sum of **p**roportions
[5]The existence and uniqueness of $\tilde{v}$ is shown in Lemma and Definition 3.21.

Implementing the described procedure for computing the water levels efficiently is important for obtaining feasible running times of our overall approach on real-world instances. In fact, this procedure is used both when reducing the graph size as described in the following section and for obtaining a feasible initial solution of our MIP as discussed in Section 3.5.2. Since the subroutine of computing the flows makes up a large part of the running time of the whole algorithm, an efficient implementation of this subroutine is particularly important.

Computing the flows from scratch for the whole graph in each iteration is highly inefficient and causes significant overhead. Instead, if one is provided the flows from the previous iteration and the nodes $u, v \in \tilde{V}_t$ that have just been joined in the previous iteration t using Algorithm 2, the flows can be updated more efficiently. It is easy to see that the excess of nodes that are neither v nor one of its successors remain unchanged. Further, it is also easy to see that $\mathrm{wtbd}(v)$ increases exactly by $\mathrm{area}(u) \cdot \mathrm{rain} + \sum_{r \in \delta^-(u)} f(r)$. This additional water is then distributed along the subgraph that is induced by v and its successors, and the new flows are added to the flows that have been computed in the previous iteration. In fact, updating the flows in this way decreases the average running time of Algorithm 3 by more than 90% compared to recomputing the flows from scratch in each iteration.

Further, the extraction of the first flooded leaf is implemented using a Fibonacci heap, which results in a speedup of about 25% on average as opposed to extracting it using a simple sorted-array implementation. Using the efficient implementation, Algorithm 3 runs fast enough to account for a negligible amount of the total running time.

Finally, we would like to point out that the iteration indices in Algorithm 3 are chosen such that all variables concerning the flows, i.e., the flows f_t themselves, the graph $\tilde{G}_t$, the area area_t, and the sets repres_t, are initialized with one, and all variables concerning the excesses of the nodes, i.e., the excesses excess_t themselves, the proportions $\hat{p}_t$, and the sum of proportions sp_t, are initialized with zero. This enforces that, in each iteration t of the algorithm, the flows f_t are the flows on the graph $\tilde{G}_t$ and the variables $(\mathrm{excess}_t(v))_{v \in \tilde{V}_t}$ store the excesses after the flows have been distributed along the graph.

3.3.1.3 Reducing the Graph Size

The size of the original graph $G_{\mathrm{or}} = (V_{\mathrm{or}}, R_{\mathrm{or}})$ or the extended graph $G_{\mathrm{or}}^{\mathrm{ex}} = (V_{\mathrm{or}}, R_{\mathrm{or}}^{\mathrm{ex}})$ is the main determinant for the size of a problem instance. In this section, we describe how the graph size can be reduced while still maintaining a realistic model of the problem described in Section 3.2.

It is worth noting that the sizes of both the original graph and the extended graph are linear in the cardinality of the node set V_{or}, which we therefore use as a natural measure of the size of these graphs. The aim of this section is to derive the *reduced graph* $G_{\mathrm{red}} = (V_{\mathrm{red}}, R_{\mathrm{red}})$ from the original graph. In fact, applying our MIP presented in the next section based on the original graph only works for unrealistically small instances. Hence, reducing the size of the graph is actually crucial in order to obtain a model that is applicable in practice. As a quick outline of this section, we provide a short summary of the ideas of our graph size reduction techniques:

1) Instead of a fixed grid size of one meter, we use a **dynamic grid size**, which means that certain parts of the terrain surface are modeled using coarser 25m or 5m grids.
2) We **remove nodes** that do not cause flow into critical locations.
3) We **contract all nodes in non-critical locations** that dispense water to critical locations into one source node.
4) We **contract adjacent nodes of similar geodesic heights**.

Before we apply these ideas, we introduce some further definitions. To model the terrain surface using a grid size of 25m, we construct the graph $G_{25} = (V_{25}, R_{25})$ from those coordinates in the DTM where both UTM coordinates are integer multiples of 25m. This works completely analogously to the construction of the original graph. The only difference is that the shape of a node in V_{25} is no longer a square with an edge length of one meter, but now a square with an edge length of 25 meters. Consequently, each node's area in G_{25} amounts to 625sqm. Also note that, for example, a building is *on* a node $v \in V_{25}$ if its shape intersects with the shape of v, which in G_{25} is a square with an edge length of 25 meters.

In the same fashion, we construct the graph $G_5 = (V_5, R_5)$ from those coordinates in the DTM where both UTM coordinates are multiples of 5m. It is important that, although the nodes in V_{25}, V_5, and V_{or} stem from the same coordinates, the sets are disjoint as the attributes of the nodes (e.g., their areas) differ.

To obtain more information about the graphs G_{25}, G_5, and G_{or}, we first assess for each node whether there are a buildings or possible actions on it. This means that, for each node $v \in \hat{V}$, where $\hat{V}$ is one of the sets V_{25}, V_5, or V_{or}, we store a set of buildings on the node, which we denote by $B(v)$, and a set of possible actions on the node, which we denote by $\mathcal{A}(v)$.

A straightforward algorithm to obtain the sets $B(v)$ of buildings and $\mathcal{A}(v)$ of actions for all nodes $v \in V$ is to loop over the nodes and, within this loop, iterate over all buildings and actions and check if the shape of the node and the shape of the building or action intersect. However, this has a horrendous running time and can be done way more efficiently using the connectivity of the shapes of buildings and actions.

For simplicity, we only present the algorithm to compute the set $B(v)$ for all nodes $v \in V$. The computation of the sets $\mathcal{A}(v)$ works similarly. For each building, we initialize a queue q containing a single node $v' \in V$ whose shape contains the coordinate of some vertex of the building's shape. Note that the coordinate of such a node can easily be computed by rounding both components of the vertex's coordinate to the next multiple of 1, 5, or 25 depending on which of the node sets the algorithm is called for.

While the queue is not empty, we take a node v from the queue and check whether it intersects with the building. If this is the case, we add the building to $B(v)$ and add the nodes north, south, west, and east of v that have not yet been processed for this building to the queue. The pseudocode is provided in Algorithm 4, which takes the set B of buildings and a node set $\hat{V} \in V_{25}, V_5, V_{\text{or}}$ as its two arguments.

1) Using a dynamic grid size: We construct a graph with a dynamic grid size, which we denote by $G_{\text{dg}} = (V_{\text{dg}}, R_{\text{dg}})$. To this end, we first construct the node set V_{dg} and then the arc set R_{dg}. To construct the node set, we initialize V_{dg} as a copy of V_{25}, then resolve each node

Algorithm 4: COMPUTE-BUILDINGS-ON-NODES

```
1 Procedure computeBuildingsOnNodes($B, \hat{V}$)
2     Initialize $B(v) = \emptyset$ for all $v \in \hat{V}$
3     for $\beta \in B$ do
4         Compute $v' \in \hat{V}$ containing an arbitrary vertex of the building's shape
5         Initialize $q = [v']$ and visited $= \emptyset$
6         while $q$ is nonempty do
7             $v = q$.pop()
8             if $\beta$ is on $v$ and $v \notin$ visited then
9                 Add the nodes north, south, west, and east of $v$ that are not in visited to $q$
10                Add $\beta$ to $B(v)$
11            Add $v$ to visited
12    return $B(v)$ for all $v \in \hat{V}$
```

that intersects with a building or an action at a 5m grid size, and finally resolve each node that has been resolved at a 5m grid size and that intersects with a ditch or an embankment at a grid size of 1m. This procedure is described in Algorithm 5, where two shapes intersect if the area of their intersection is strictly positive. To keep track of the resolution at the single nodes, the resolution $\text{res}(v)$ is stored for each node $v \in V_{\text{dg}}$.

Algorithm 5: CONSTRUCT-NODES

```
1 Procedure constructNodes($V_{25}, V_5, V_{\text{or}}$)
2     Initialize $V_{\text{dg}} = V_{25}$, queue $= \emptyset$, and $\text{res}(v) = 25$ for all $v \in V_{\text{dg}}$
3     for $v \in V_{25}$ do
4         if $v$ intersects with a building or an action then
5             Replace $v$ in $V_{\text{dg}}$ by the nodes in $V_5^v \subseteq V_5$, where $V_5^v$ is the set of nodes
                in $V_{25}$ whose shapes intersect with the shape of $v$ (i.e., resolve this the
                node $v$ at a 5m grid size). For each $v' \in V_5^v$, set $\text{res}(v') := 5$.
6             Insert all newly added nodes into the queue
7     for $v \in$ queue do
8         if there is a ditch or embankment on $v$ then
9             Replace $v$ in $V_{\text{dg}}$ by the nodes in $V_{\text{or}}^v \subseteq V_{\text{or}}$, where $V_{\text{or}}^v$ is the set of nodes
                in $V_{\text{or}}$ whose shapes intersect with the shape of $v$ (i.e., resolve the node $v$ at
                a 1m grid size). For each $v' \in V_{\text{or}}^v$, set $\text{res}(v') := 1$.
10    return $V_{\text{dg}}$
```

The set of arcs R_{dg} is then constructed by adding an arc between two nodes in V_{dg} if and only if they are adjacent on the dynamic grid (i.e., their shapes have a common edge). The arc is again directed from the node with the higher index to the node with the lower index according to the ordering of the corresponding nodes with the same coordinates in V_{or} (i.e., from the higher node to the lower node whenever the two nodes do not have the same geodesic height).

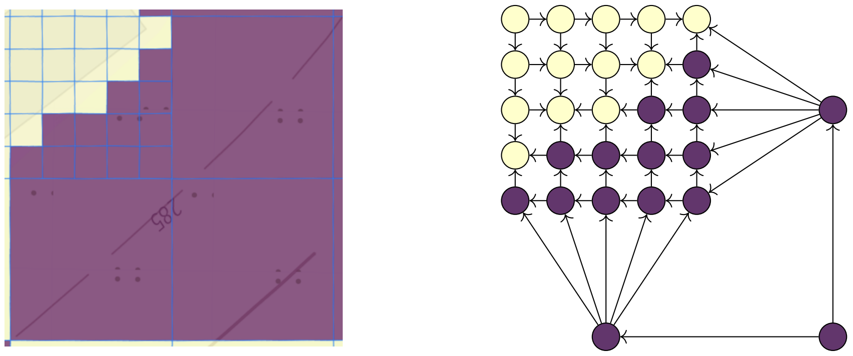


Figure 3.4.: A screenshot from our web application on the left-hand side, where the dynamic grid size is visualized and the nodes intersecting with a building are colored yellow. The corresponding part of the graph $G_{\text{dg}} = (V_{\text{dg}}, R_{\text{dg}})$ is visualized on the right-hand side.

To compute the ratios, we also have to take the resolutions of the nodes into account. This stems from the fact that, with the dynamic grid size, the length of the common edge of two adjacent nodes' shapes can be 1 meter, 5 meters, or 25 meters. The ratio of an arc $r \in R_{\text{dg}}$, hence, depends on the slopes of the outgoing arcs of $\alpha(r)$ and on the proportion of the boundary of the shape of $\alpha(r)$ that the shapes of $\alpha(r)$ and $\omega(r)$ have in common. The ratio of r is computed as

$$\text{ratio}(r) := \left({}^{\text{slope}(r)}\!/\!{}_{\sum_{\hat{r}\in\delta^+(\alpha(r))}\text{slope}(\hat{r})}\right) \cdot \text{correction}(r),$$

where

$$\text{correction}(r) := \begin{cases} 1 & \text{if } \text{res}(\alpha(r)) \leqslant \text{res}(\omega(r)) \\ {}^{\text{res}(\omega(r))}\!/\!{}_{\text{res}(\alpha(r))} & \text{else.} \end{cases}$$

An example of shapes of nodes and the corresponding graph $G_{\text{dg}} = (V_{\text{dg}}, R_{\text{dg}})$ is provided in Figure 3.4.

Modeling all buildings at a grid size of five meters is still overly exact. Buildings at which no (or only negligible) water levels are to be expected can still be modeled at a grid size of 25m. To assess a good grid size, we compute the water levels on the graphs $G_{\text{dg}} = (V_{\text{dg}}, R_{\text{dg}})$ and $G_{25} = (V_{25}, R_{25})$ using Algorithm 3, which has been presented previously.

We call a node $v \in V_{25}$ *threatened*, if it has a strictly positive water level in the computation on $G_{25} = (V_{25}, R_{25})$ or if any node in V_{dg} whose shape intersects with the shape of v has a water level greater than or equal to one centimeter in the computation on $G_{\text{dg}} = (V_{\text{dg}}, R_{\text{dg}})$.

For each non-threatened node $v \in V_{25}$ that only intersects with buildings and not with actions, we rescale its resolution in $G_{\text{dg}} = (V_{\text{dg}}, R_{\text{dg}})$ back to 25 meters, i.e., we contract all nodes in V_{dg} whose shapes intersect with the shape of v into v. Afterwards, we recompute the arc set R_{dg} and the ratios with the updated node set V_{dg} as we have done before, which yields the final version of $G_{\text{dg}} = (V_{\text{dg}}, R_{\text{dg}})$.

The reduction in the overall number of nodes achieved by this step highly depends on the number of nodes in V_{or} that do not intersect with any buildings or actions, as the number of these nodes is reduced by the highest factor of 625. In the instances presented in Section 3.5, the overall number of nodes is usually reduced by a factor of about 500.

2) Removing nodes not causing flow into critical locations: Our next goal is to remove nodes from the graph that do not cause any flow into critical locations. To this end, we define four new properties for nodes. A node $v \in V_{\text{dg}}$ is called …

- *critical* if its shape intersects with a building or an action.
- *relevant* if it is critical, its resolution is not 25m, or it is a successor of a critical node in $G_{\text{dg}} = (V_{\text{dg}}, R_{\text{dg}})$. Apart from critical nodes, relevant nodes are either nodes where water may accumulate and then cause critical nodes to be flooded due to back pressure, or nodes that are needed to complete the grid without gaps.
- *water-dispensing* if it is not relevant, but it is a predecessor of a relevant node in the graph $G_{\text{dg}} = (V_{\text{dg}}, R_{\text{dg}})$. Water accumulating on such nodes does not cause flooding of relevant nodes due to back pressure. These nodes are, however, still interesting as they dispense water to relevant nodes.
- *irrelevant* if it is neither of the above. Irrelevant nodes do not contribute in any way to the flooding of relevant nodes.

As an example, think of a municipality at the foot of a mountain. Here, the nodes at coordinates within the municipality are the relevant nodes, the nodes at coordinates on the side of the mountain facing the municipality are the water-dispensing nodes, and the nodes at coordinates on sides of the mountain not facing the municipality are the irrelevant nodes.

The first step of the node removal consists of removing all irrelevant nodes from V_{dg}. It is worth noting that this may cause the graph to be no longer weakly connected. In practice though, this only happens if buildings are spread far apart, which is seldom the case. Apart from this, our model still works if the graph is not weakly connected. We denote the graph obtained by this method by $G_{\text{ri}} = (V_{\text{ri}}, R_{\text{ri}})$.[6]

The reduction in the overall number of nodes achieved by this step highly depends on the number of irrelevant nodes, which in turn depends on the choice of the input DTM. Barely any nodes are irrelevant in cases where the region covered by the DTM is chosen relatively tight around the build-up region to be protected, whereas a lot of nodes are irrelevant if the region covered by the DTM is chosen relatively large. However, since the region covered by the DTM is composed of one by one kilometer rectangles and must always be chosen large enough so that no potentially relevant or water-dispensing nodes are omitted, a certain number of irrelevant nodes is usually unavoidable, so the removal of irrelevant nodes represents an important step in reducing the overall number of nodes.

3) Contracting nodes in non-critical locations: In the next step, we deal with the water-dispensing nodes. By construction, flow through these nodes is not affected by the decision on which actions are taken. To reduce the graph size, we contract all water-dispensing nodes

[6]**ri**: **r**emove **i**rrelevant.

into a single node s, which we call the *source node*. The obtained graph is denoted by $G_{\text{wd}} = (V_{\text{wd}}, R_{\text{wd}})$.[7] Note that this contraction also changes the arc set. The arcs that are incident to s arise from arcs in G_{ri} that are directed from a water-dispensing node to a relevant node. In particular, this means that the in-degree of s is zero. Further, this construction might lead to parallel arcs. To keep the graph as small as possible, parallel arcs starting in s are contracted into a single arc.

The area of the source node is set to the sum of the areas of all water-dispensing nodes. To compute the ratios of the arcs that are incident to s, we first compute the flows in the graph G_{ri} using Algorithm 1 and denote the resulting flow on $r \in R_{\text{ri}}$ by $f(r)$. The ratio of an arc $r \in R_{\text{wd}}$ starting in s is then set to the sum of the inflow into $\omega(r)$ from water-dispensing nodes divided by the total inflow from water-dispensing into relevant nodes in G_{ri}:

$$\text{ratio}(r) := \sum_{\substack{\hat{r} \in R_{\text{ri}}: \\ \alpha(\hat{r}) \text{ is water-dispensing} \\ \text{and } \omega(\hat{r}) = \omega(r)}} f(\hat{r}) \Bigg/ \sum_{\substack{\tilde{r} \in R_{\text{ri}}: \\ \alpha(\tilde{r}) \text{ is water-dispensing} \\ \text{and } \omega(\tilde{r}) \text{ is relevant}}} f(\tilde{r})$$

For completeness, we set the geodesic height of s to the largest geodesic height in the graph before contraction plus one meter. This ensures that the source node is never flooded unless an unrealistically large amount of rain per sqm is used.

4) Contracting adjacent nodes of similar geodesic heights: As a last step, we contract adjacent nodes into a new node if they have the same geodesic height up to a given threshold and the same combination of actions and buildings on them, which yields the desired *reduced graph* $G_{\text{red}} = (V_{\text{red}}, R_{\text{red}})$. The exact procedure for computing G_{red} is presented in Algorithm 6, which will be explained in the following paragraphs. The corresponding reduction step has two benefits. First, it further reduces the number of nodes. Second, and far more beneficially, it greatly improves the numerical stability of the MIP. Indeed, numerical issues caused the MIP to be infeasible before we introduced this procedure. The improved numerical stability stems from the fact that, after the procedure, all nodes in the resulting reduced graph G_{red} have pairwise distinct geodesic heights, and there are only few adjacent nodes that have similar geodesic heights.

Algorithm 6 is divided into four parts. In the first part, we contract adjacent nodes that intersect with the same set of buildings and actions and have a similar geodesic height into a new node representing the contracted nodes. The shape of such a new node is defined as the union of the shapes of the contracted nodes, and the boundary of such a node is the boundary of its shape. The geodesic height of the new node is then set to the area-weighted average over the geodesic heights of the nodes that have been contracted into the new node $v \in V_{\text{red}}$, i.e., it is set as follows:

$$\text{gh}(v) := \sum_{\substack{v' \in V_{\text{wd}}: \\ v' \text{ is contracted into } v}} \text{gh}(v') \cdot \text{area}(v') \Bigg/ \sum_{\substack{v' \in V_{\text{wd}}: \\ v' \text{ is contracted into } v}} \text{area}(v')$$

[7] **wd**: **w**ater **d**ispensing

Algorithm 6: CONTRACT-COMPONENTS

```
1  **Procedure** contractComponents($G_{\text{wd}} = (V_{\text{wd}}, R_{\text{wd}}), \text{threshold}, \varepsilon$)
2      Initialize $G_{\text{red}} = (V_{\text{red}}, R_{\text{red}})$ as a copy of $G_{\text{wd}} = (V_{\text{wd}}, R_{\text{wd}})$
3      For each node $v \in V_{\text{red}}$, compute its geodesic height rounded to a multiple of the threshold
         and store it in $\text{rgh}(v)$
4      Initialize $\text{original\_nodes}(v) = \{v\}$ for all $v \in V_{\text{red}}$
5      # Contract nodes
6      **while** *There are adjactent nodes* $u, v \in V_{\text{red}}$ *with* $\text{rgh}(v) = \text{rgh}(u)$ *and the buildings and
         actions on both nodes are the same* **do**
7          Contract $u, v$ into $\hat{v}$ and set
             $\text{original\_nodes}(\hat{v}) = \text{original\_nodes}(u) \cup \text{original\_nodes}(v)$
8      **for** $v \in V_{\text{red}}$ **do**
9          Set $\text{gh}(v) := \sum_{v' \in \text{original\_nodes}(v)} \text{gh}(v') \cdot \text{area}(v') / \sum_{v' \in \text{original\_nodes}(v)} \text{area}(v')$
10     # Enforce pairwise distinct geodesic heights
11     **while** *There are two nodes* $u, v \in V_{\text{red}}$ *with* $\text{gh}(u) = \text{gh}(v)$ **do**
12         **if** $\text{gh}(u) = \min_{v' \in V_{\text{red}}} \text{gh}(v')$ **then**
13             Set $\text{gh}(v) = \text{gh}(v) - \varepsilon$
14         **else**
15             Let $v' \in V_{\text{red}}$ with $\text{gh}(v')$ maximal such that $\text{gh}(v') < \text{gh}(v)$
16             Set $\text{gh}(v) = \text{gh}(v) - \min\{\varepsilon, \frac{1}{2} \cdot (\text{gh}(v) - \text{gh}(v'))\}$
17     # Remove uphill arcs
18     **for** $r \in R_{\text{red}}$ **do**
19         **if** $\text{gh}(\alpha(r)) < \text{gh}(\omega(r))$ **then**
20             Remove $r$ from $R_{\text{red}}$ and add $\overleftarrow{r}$ to $R_{\text{red}}$ if it does not already exist
21     # Recompute ratios
22     **for** $v \in V_{\text{red}}$ **do**
23         $\text{sum\_of\_slopes}(v) = \sum_{r \in \delta^+_{G_{\text{red}}}(v)} \text{slope}(r)$
24         Compute the length of the boundary of $v$ in meters and store the value in
             $\text{length\_of\_boundary}(v)$
25         **for** $r \in \delta^+_{G_{\text{red}}}(v)$ **do**
26             Compute the length of the intersection of the boundaries of $v$ and $\omega(r)$ and store
                 the value in $\text{common\_boundary}(r)$
27             $\text{ratio}(r) =$
                 $(\text{slope}(r) \cdot \text{common\_boundary}(r)) / (\text{sum\_of\_slopes}(v) \cdot \text{length\_of\_boundary}(v))$
28     # Remove $s$
29     **for** $r \in \delta^+_{G_{\text{red}}}(s)$ **do**
30         $\text{area}(\omega(r)) = \text{area}(\omega(r)) + \text{area}(s) \cdot \text{ratio}(r) / \sum_{r' \in \delta^+_{G_{\text{red}}}(s)} \text{ratio}(r')$
31     Remove $s$ and all its incident arcs from $G_{\text{red}}$
32     **return** $G_{\text{red}}$
```

In practice, this procedure usually leads to all nodes in V_{red} having pairwise distinct geodesic heights. However, if this is not the case, we add a slight noise to the geodesic heights of each pair of nodes that have the same geodesic height. This is important in order to guarantee that the MIP produces a feasible solution of the problem. It is worth noting that DTMs are usually only exact up to some centimeters and, hence adding this noise is within the measuring tolerance.
In the second part, we remove uphill arcs $r \in R_{\text{red}}$ that might arise during this procedure and add the corresponding reversed arcs if they do not already exist.

In the third part, we recompute the ratios of the newly obtained arcs. This time, for a node $v \in V_{\text{red}}$, the ratio of an arc $r \in \delta^{+}_{G_{\text{red}}}(v)$ is set proportionally to the slopes of the arcs leaving v and to the length of the intersection of the boundaries of v and $\omega(r)$.

In the final part, we remove the node s and instead increase the area of nodes that are adjacent to s. This only decreases the size of the graph by a single node, but, again, greatly improves the numerical stability of the MIP.

Finding a good value for the threshold is critical here. An overly high value leads to unrealistic results, whereas an overly low value decreases the performance gain obtained from the contraction. From our experience on numerous real-world instances from the project AKUT, depending on the terrain surface, we recommend choosing a value between 5cm and 15cm. On hilly surfaces, the value can preferably be set a bit higher, whereas on smooth surfaces, it is better to stick to low values.

The reduction in the overall number of nodes achieved in this last step mainly depends on the threshold parameter and the hilliness of the modeled region. The higher the threshold parameter and the flatter the region, the greater the reduction in the number of nodes.

For three representative regions, which are revisited later in Section 3.5.2, an overview of the reduction in the overall number of nodes from G_{or} to G_{red} is provided in Table 3.2.

Region	$\lvert V_{\text{or}} \rvert$	$\lvert V_{\text{red}} \rvert$	Factor
Hilly Region	12,239,475	4719	2594
Flat Region 1	2,523,799	6613	382
Flat Region 2	1,789,498	3778	474

Table 3.2.: Reduction in the total number of nodes achieved for three representative regions, where the factor provided in the third column is obtained as $|V_{\text{or}}|/|V_{\text{red}}|$.

The *extended reduced graph* $G^{\text{ex}}_{\text{red}} = (V^{\text{ex}}_{\text{red}}, R^{\text{ex}}_{\text{red}})$ is constructed from the reduced graph $G_{\text{red}} = (V_{\text{red}}, R_{\text{red}})$ returned by Algorithm 6 in the same manner as we constructed it for the original graph, i.e., for each arc $r \in R_{\text{red}}$, we add a copy of r in reverse direction.

3.3.2 Mixed-Integer Programming Formulation and Presolve Techniques

In this section, we present our mixed-integer programming formulation of the problem defined in Section 3.3.1 as well as several intuitive valid inequalities that improve the performance. The constraints are formulated verbally, while the mathematical formulation can be found in

Appendix A.1. Afterwards, we further describe methods to preset some of the variables, which is important to obtain feasible running times.

3.3.2.1 Mixed-Integer Programming Formulation

Before stating the mixed-integer programming formulation, we provide complete lists of the sets, parameters, and variables for better readability. The MIP takes, among other input data, a graph and its extended graph as an input. Any of the graphs that are presented in the previous section can be used, but, as already mentioned, we highly recommend using the reduced graph (and the corresponding extended reduced graph) here as all other graphs make the model too large or numerically unstable to be applied to realistic instances. The graph used in the MIP is denoted by $G = (V, R)$ and the corresponding extended graph by $G^{\text{ex}} = (V, R^{\text{ex}})$. Throughout this section, we assume that the nodes in V have pairwise distinct geodesic heights, which is the case if $G = G_{\text{red}}$.

Sets:

V	node set of the graph
R	arc set of the graph
R^{ex}	arc set of the extended graph
B	set of buildings
$\mathcal{B}$	set of possible retention basins
$\mathcal{D}$	set of possible ditches
$\mathcal{E}$	set of possible embankments
$\mathcal{A}$	set of all possible actions, where $\mathcal{A} = \mathcal{B} \cup \mathcal{D} \cup \mathcal{E}$
P	set of properties
$P_{\text{yellow}} \subseteq P$	set of properties where the corresponding actor needs minor incentives to cooperate
$P_{\text{red}} \subseteq P$	set of properties where the corresponding actor needs major incentives to cooperate
$P_{\text{black}} \subseteq P$	set of properties where the corresponding actor does not cooperate at all

The sets corresponding to possible actions are denoted by calligraphic letters. We further introduce the set $\mathcal{B}(v) \subseteq \mathcal{B}$ for each $v \in V$ as the set of basins on v. The sets $\mathcal{D}(v)$ and $\mathcal{E}(v)$ are defined analogously, and we let $V(\beta)$ denote the set of all nodes intersecting with building $\beta \in B$.

Parameters:

rain	total rain per sqm in m
budget	budget for the total cost of taken actions
$\text{GH}(v)$	original geodesic height of node $v \in V$
$\text{area}(v)$	area of node $v \in V$ in sqm
$\text{ratio}(r)$	ratio of outflow of node $\alpha(r)$ allocated to arc $r \in R^{\text{ex}}$
$\text{depth}(a)$	depth of basin or ditch $a \in \mathcal{B} \cup \mathcal{D}$ in m
$\text{height}(e)$	height of embankment $e \in \mathcal{E}$ in m
$\text{cost}(a)$	cost of action $a \in \mathcal{A}$

thresholdWL(k)	threshold water level in m for hazard class $k \in \{0, 1, 2, 3\}$
damage(k, β)	damage in the objective function if building $\beta \in B$ belongs to hazard class $k \in \{1, 2, 3, 4\}$
maxAllowedYellow	maximum number of properties needing minor incentives to cooperate that actions can be built on
maxAllowedRed	maximum number of properties needing major incentives to cooperate that actions can be built on

Variables:

$f(r)$	total flow on arc $r \in R^{\text{ex}}$ in m^3
excess(v)	excess of node $v \in V$ in m^3
wl(v)	water level at node $v \in V$ in m
flooded(v)	1 if wl$(v) > 0$ at node $v \in V$, 0 otherwise
active(r)	1 if there is flow along arc $r \in R^{\text{ex}}$, 0 otherwise
full(r)	1 if wl$(\alpha(r)) > 0$ for $r \in R$, 0 otherwise
decBasin(b)	1 if basin $b \in \mathcal{B}$ is built, 0 otherwise
decDitch(d)	1 if ditch $d \in \mathcal{D}$ is built, 0 otherwise
decEmb(e)	1 if embankment $e \in \mathcal{E}$ is built, 0 otherwise
gh(v)	geodesic height of node $v \in V$ after actions have been built in m
down(v)	1 if a ditch or basin is built on $v \in V$, 0 otherwise
max_inc(v)	maximum increase of height through building embankments on $v \in V$ in m
max_dec(v)	maximum decrease of height through building ditches or basins on $v \in V$ in m
aux_fd(r)	binary auxiliary variable for the flow distribution over arc $r \in R^{\text{ex}}$: 1 if arc is active and not full, 0 otherwise
od(r)	1 if node $\alpha(r)$ is higher than node $\omega(r)$ after building the actions for $r \in R$, 0 otherwise
auxO1F1(r)	binary auxiliary variable for $r \in R$: 1 if od$(r) = 1$ and full$(r) = 1$, 0 otherwise
auxO1F0(r)	binary auxiliary variable for $r \in R$: 1 if od$(r) = 1$ and full$(r) = 0$, 0 otherwise
auxO0F1(r)	binary auxiliary variable for $r \in R$: 1 if od$(r) = 0$ and full$(r) = 1$, 0 otherwise
auxO0F0(r)	binary auxiliary variable for $r \in R$: 1 if od$(r) = 0$ and full$(r) = 0$, 0 otherwise
max_wl(β)	maximum water level at any node intersecting with building $\beta \in B$ in m
hc(k, β)	1 if building $\beta \in B$ belongs to hazard class $k \in \{0, \ldots, 4\}$, 0 otherwise
action(p)	1 if an action is taken on property $p \in P$, 0 otherwise
hdb(b)	depth of basin $b \in \mathcal{B}$ in m if basin b is built, 0 otherwise
hdd(d)	depth of ditch $d \in \mathcal{D}$ in m if ditch d is built, 0 otherwise
hde(e)	height of embankment $e \in \mathcal{E}$ in m if it is built, 0 otherwise

Objective function:

The only term in the objective function is the damage occurring at the buildings, which depends on their hazard class and their damage class.[8] Thus, the objective function to be minimized is given as

$$\sum_{\beta \in B} \sum_{k=1}^{4} \mathrm{damage}(k, \beta) \cdot \mathrm{hc}(k, \beta).$$

Constraints:
To enhance readability, we use the $\max$ operator within our formulation. This operator takes a set of variables and / or parameters as an argument and returns the maximum among their values. Note that the operator can alternatively be implemented using big M constraints. This, however, may lead to numerical instability if finding a suitable value M is difficult. We therefore use the $\max$ operator, which is pre-implemented in most modern mixed-integer programming solvers.

Furthermore, we make use of indicator constraints. An indicator constraint is of the form

$$bin = val \quad \Longrightarrow \quad a^T x \leqslant b$$

and states that the constraint $a^T x \leqslant b$ must be satisfied if the binary variable bin has value $val \in \{0, 1\}$. An indicator constraint can also be implemented using a big M constraint. It is, however, well-known that indicator constraints have many advantages compared to big M formulations [Bon+15]. Indicator constraints are, like the $\max$ operator, pre-implemented in many modern mixed-integer programming solvers.

The formulation of some constraints requires using strict inequalities, which is not possible theoretically in a MIP. In practice, however, values are encoded as floats with a bounded number of decimal places. Therefore, a strict inequality $x < y$ can be formulated as $x \leqslant y - \varepsilon$ for some small $\varepsilon > 0$.

Water levels at nodes: To determine the water levels, we first compute the excess of each node $v \in V$:

(1) The excess of node $v \in V$ is the inflow minus the outflow plus the rain volume on the node.

The excess of a node $v \in V$ immediately yields the water level at the node:

(2) The water level at node $v \in V$ is the excess of node v divided by its area.

Geodesic heights of nodes: In contrast to most traditional flow problems, we do not aim to optimize the flow in the graph, but the terrain surface determining the flows. The following constraints therefore set the geodesic height variable $\mathrm{gh}(v)$ for each node $v \in V$. First, to distinguish the two cases (GH1) and (GH2) from Section 3.3.1.2, the variable $\mathrm{down}(v)$ is set to one in case (GH1), i.e., if a basin or ditch is built on the node, and to zero otherwise:

(3) If a basin $b \in \mathcal{B}$ is built on node $v \in V$, the variable $\mathrm{down}(v)$ is set to one.

[8] Since buildings of hazard class 0 do not contribute to the objective function, we only have to sum k from 1 to 4.

(4) If a ditch $d \in \mathcal{D}$ is built on node $v \in V$, the variable down(v) is set to one.

(5) If neither ditches nor basins are built on node $v \in V$, the variable down(v) is set to zero.

Next, the variables hdb(b), hdd(d), and hde(e) for $b \in \mathcal{B}$, $d \in \mathcal{D}$, and $e \in \mathcal{E}$ that determine the height differences that result from taking actions are set:

(6) The variable hdb(b) is set to depth(b) if basin $b \in \mathcal{B}$ is built (i.e., if decBasin$(b) = 1$), and to zero otherwise.

(7) The variable hdd(d) is set to depth(d) if ditch $b \in \mathcal{B}$ is built (i.e., if decDitch$(d) = 1$), and to zero otherwise.

(8) The variable hde(e) is set to height(e) if embankment $e \in \mathcal{E}$ is built (i.e., if decEmb$(e) = 1$), and to zero otherwise.

To enable setting the geodesic height variables as described in the case distinction, the maximum depth of any of the basins or ditches built on v in case (GH1) and the maximum height of any of the embankments built on v in case (GH2) is now computed:

(9) The maximum decrease max_dec(v) of the geodesic height at node $v \in V$ is set to the maximum of the height differences that result from building basins or ditches on v and 0.

(10) The maximum increase of the geodesic height max_inc(v) at node $v \in V$ is set to the maximum of the height differences that result from building embankments on v and 0.

Finally, the geodesic height variable gh(v) is set for each node $v \in V$:

(11) The geodesic height gh(v) of node $v \in V$ is greater than or equal to the original geodesic height of v minus the maximum decrease caused by basins and ditches.

(12) The geodesic height gh(v) of node $v \in V$ is less than or equal to the original geodesic height of v plus the maximum increase caused by embankments.

(13) If a basin or ditch is built on node $v \in V$ (i.e., down$(v) = 1$), the geodesic height gh(v) of v is less than or equal to the original geodesic height of v minus the maximum decrease caused by basins and ditches and, hence, in combination with Constraint (11), equal to the original geodesic height of v minus the maximum decrease caused by basins and ditches. This is modeled using a big M constraint where $M(v) := \max(\{\text{depth}(b) | b \in \mathcal{B}(v)\} \cup \{\text{depth}(d) | d \in \mathcal{D}(v)\} \cup \{0\}) + \max(\{\text{height}(e) | e \in \mathcal{E}(v)\} \cup \{0\})$.

(14) If no basin or ditch is built on node $v \in V$ (i.e., down$(v) = 0$), the geodesic height gh(v) of v is greater than or equal to the original geodesic height of v plus the maximum increase caused by embankments and, hence, in combination with Constraint (12), equal to the original geodesic height of v plus the maximum increase caused by embankments. This is again modeled using a big M constraint with the same $M(v)$ as in the previous constraint.

Arc directions: There might be arcs in the input graph where, after taking actions and thereby changing the geodesic heights of nodes, the start node has a lower geodesic height than the end node, so the direction of the arc has to be reversed. If this is *not* the case for an arc $r \in R$, the arc is said to have *original direction* and the variable $\text{od}(r)$ is set to one by using indicator constraints:

(15) If arc $r \in R$ has original direction, the variable $\text{od}(r)$ is set to one.

(16) Otherwise, the variable $\text{od}(r)$ is set to zero.

Full arcs: The following constraints deal with the behavior of the flows on the arcs in the extended graph $G^{\text{ex}} = (V, R^{\text{ex}})$. To this end, we introduce the following terminology: An arc $r \in R$ is called *full* if the water level at the lower of the two nodes $\alpha(r)$ and $\omega(r)$ is greater than or equal to the absolute difference of their geodesic heights. For its inverse arc $\overleftarrow{r} \in R^{\text{ex}} \backslash R$, we say that this arc is full if and only if r is full.[9] Note that this definition refers to the geodesic heights after taking actions, where it is possible that $\alpha(r)$ has a smaller geodesic height than $\omega(r)$. To connect the variables $\text{full}(r)$ to the water levels, some binary auxiliary variables incorporating the original direction variables are first introduced:

(17) The variable $\text{auxO1F1}(r)$ for arc $r \in R$ is set to one if and only if $\text{od}(r) = 1$ and $\text{full}(r) = 1$.

(18) The variable $\text{auxO1F0}(r)$ for arc $r \in R$ is set to one if and only if $\text{od}(r) = 1$ and $\text{full}(r) = 0$.

(19) The variable $\text{auxO0F1}(r)$ for arc $r \in R$ is set to one if and only if $\text{od}(r) = 0$ and $\text{full}(r) = 1$.

(20) The variable $\text{auxO0F0}(r)$ for arc $r \in R$ is set to one if and only if $\text{od}(r) = 0$ and $\text{full}(r) = 0$.

The following constraints connect the variables $\text{full}(r)$ to the water levels using the auxiliary variables:

(21) If arc $r \in R$ has original direction and is full, the water level at $\omega(r)$ must be greater than or equal to the absolute difference of the geodesic heights of $\alpha(r)$ and $\omega(r)$.

(22) If arc $r \in R$ has original direction and is not full, the water level at $\omega(r)$ must be less than the absolute difference of the geodesic heights of $\alpha(r)$ and $\omega(r)$.

(23) If arc $r \in R$ does not have original direction and is full, the water level at $\alpha(r)$ must be greater than or equal to the absolute difference of the geodesic heights of $\alpha(r)$ and $\omega(r)$.

[9]Nodes in G_{red} have pairwise distinct geodesic heights, so the lower node is always well-defined. In practice, the geodesic heights after taking actions are also pairwise distinct. If this is not the case, one can decrease the depth of height of the action that causes the issue by a small value similarly to how pairwise distinct geodesic heights of nodes are enforced in G_{red}.

(24) If arc $r \in R$ does not have original direction and is not full, the water level at $\alpha(r)$ must be less than the absolute difference of the geodesic heights of $\alpha(r)$ and $\omega(r)$.

Flooded nodes: A node $v \in V$ is called *flooded* if its water level $\mathrm{wl}(v)$ is strictly positive, and *non-flooded* otherwise. The following indicator constraints set the variables $\mathrm{flooded}(v)$ for $v \in V$ indicating which nodes are flooded:

(25) If the water level $\mathrm{wl}(v)$ at node $v \in V$ is strictly positive, the variable $\mathrm{flooded}(v)$ is set to one.

(26) If the water level $\mathrm{wl}(v)$ at node v is zero, the variable $\mathrm{flooded}(v)$ is set to zero.

Active arcs: The net flow between two adjacent nodes in the extended graph can be in either one or the other direction. An arc $r \in R^{\mathrm{ex}}$ is called *active* if the flow on r is strictly positive. The following constraints set the variables $\mathrm{active}(r)$ for $r \in R^{\mathrm{ex}}$ that indicate active arcs:

(27) For arc $r \in R$ and its inverse arc $\overleftarrow{r} \in R^{\mathrm{ex}}$, at most one of the variables $\mathrm{active}(r)$ and $\mathrm{active}\left(\overleftarrow{r}\right)$ can be equal to one.

(28) If an arc $r \in R^{\mathrm{ex}}$ is not active, the flow on the arc must be zero.

Flow on arcs that are not full: The outflow of a node $v \in V$ is to be distributed according to the ratios of its outgoing arcs in the extended graph $G^{ex} = (V, R^{\mathrm{ex}})$ that are active and not full. The following constraints set the auxiliary variables $\mathrm{aux\_fd}(r)$ and $\mathrm{aux\_fd}\left(\overleftarrow{r}\right)$ for $r \in R$ that indicate arcs that are both active and full:

(29) For arc $r \in R$, the auxiliary variable $\mathrm{aux\_fd}(r)$ is set to one if and only if the arc is active and not full.

(30) For arc $r \in R$, the auxiliary variable $\mathrm{aux\_fd}\left(\overleftarrow{r}\right)$ for the inverse arc is set to one if and only if $\overleftarrow{r}$ is active and not full.[10]

The outflow of each node $v \in V$ is now distributed among its outgoing arcs in the extended graph that are active and not full:

(31) For node $v \in V$ and each pair of arcs $r_1, r_2 \in \delta^+_{G^{\mathrm{ex}}}(v)$, if both arcs are active and not full, the flow is distributed proportionally to the ratios $\mathrm{ratio}(r_1)$ and $\mathrm{ratio}(r_2)$.

For each arc $r \in R$ that is not full, the water level at the higher of the two nodes $\alpha(r)$ and $\omega(r)$ must be zero.

(32) For each arc $r \in R$ that is not full and has original direction, the water level at $\alpha(r)$ is set to zero.

(33) For each arc $r \in R$ that is not full and does not have original direction, the water level at $\omega(r)$ is set to zero.

[10]Recall that $\overleftarrow{r}$ is full if and only if r is full.

For a non-full arc $r \in R$, water can only flow in downhill direction:

(34) For each arc $r \in R$ that is not full and has original direction, the arc $\overleftarrow{r}$ is not active.

(35) For each arc $r \in R$ that is not full and does not have original direction, the arc r is not active.

Flow on full arcs: As the flow is immediately connected to the water levels by Constraints (1) and (2), the flow on each full arc $r \in R$ can be set indirectly by connecting the water levels at its start node and its end node:

(36) For each full arc $r \in R$, the sum of the geodesic height and the water level must be equal in $\alpha(r)$ and $\omega(r)$.

Maximum water levels at buildings:

(37) For each building $\beta \in B$, the maximum water level variable max_wl(β) is set to the maximum of the water levels at nodes intersecting with the building.

Note that, strictly speaking, the maximum is not taken here, but the maximum water level at the building is only bounded from below by each water level at an intersecting node. The objective function then aims to minimize the maximum water levels at the buildings to achieve equality.

Hazard classes of buildings:

(38) Each building $\beta \in B$ belongs to exactly one hazard class.

(39) If building $\beta \in B$ belongs to hazard class $k \in \{0, \dots, 4\}$, its maximum water level must be less than or equal to the upper threshold of this hazard class.

Again, the maximum water levels are only bounded from above as a higher hazard class leads to a higher penalty in the objective function.

Budget constraint:

(40) The total cost for building basins, ditches, and embankments must not exceed the given budget.

Incentives for actors: The following constraints enforce the given upper bounds on the incentives required for cooperation of actors and ensure that no actions are taken on properties of actors that do not cooperate at all. This is done by means of the variables action(p) for $p \in P$ that indicate properties on which at least one action is taken:

(41) Actions are taken on at most maxAllowedYellow + maxAllowedRed yellow and red properties in total.

(42) Actions are taken on at most maxAllowedRed red properties.

(43) No actions are taken on black properties.

(44) The variable action(p) for property $p \in P$ is set to one if at least one action is taken on property p.

It is worth noting that it is not trivial to see that the MIP is indeed a correct formulation of the problem defined in Section 3.3.1. However, we show this in Section 3.4 by proving that (1) for every set $D \subseteq \mathcal{A}$ that satisfies Constraints (40)-(44), there exists a feasible solution taking exactly the actions in D, and that (2) any feasible solution of the MIP taking exactly the actions in $D \subseteq \mathcal{A}$ leads to the same water levels at the nodes as the result of Algorithm 3 applied on G^D, which is the graph that results from taking the actions in D and adjusting the geodesic heights and arc directions accordingly as described in Section 3.2.

Valid inequalities:
We finish the description of the MIP by presenting three intuitive sets of valid inequalities that improve the solution times of the model:

(45) For each pair of consecutive original-direction (i.e., downhill) arcs $r_1, r_2 \in R$ with $\omega(r_1) = \alpha(r_2)$, the first arc r_1 can only be full if the second arc r_2 is full as well.

(46) If node $v \in V$ is flooded, then each arc $r \in \delta^+_{G^{\text{ex}}}(v)$ with $\text{gh}(v) > \text{gh}(\omega(r))$ must be full (otherwise, water could still flow in downhill direction from v).

(47) If node $v \in V$ is not flooded, then no arc $r \in \delta^-_{G^{\text{ex}}}(v)$ with $\text{gh}(v) < \text{gh}(\alpha(r))$ can be full.

3.3.2.2 Presolve Techniques

We close this chapter by presenting two methods to preset some of the variables. Through our analysis, we found that the variables flooded(v) for $v \in V$ are the major bottleneck of the MIP. It is therefore natural to investigate which nodes must always be flooded and which nodes can never be flooded in a feasible solution in order to preset some of these variables to one or zero, respectively.

We start by presetting variables for nodes that must always be flooded. To this end, we consider the leaves of the graph $G = (V, R)$. If there is no possible embankment on a leaf $l \in V$ and no possible ditches or basins on any of the nodes in $\delta^-(l)$, the leaf will also be a leaf after taking actions – independent of which actions are selected. This means that l is flooded in any feasible solution since at least the initial water from the rain event will build up a water level strictly larger than zero at l. For all such leaves, we can, therefore, preset the variable flooded(l) to one.

Identifying nodes $v \in V$ for which the variable flooded(v) can be preset to zero (i.e., nodes that can never be flooded in any feasible solution) is more involved. The idea here is that, if no possible action is located on v, the water levels at all successors of v must be equal to the absolute difference of their geodesic height and the geodesic height of v in order for v to be flooded. Thus, if the total amount of rain on the whole area does not suffice for raising the water level at each successor to the absolute difference of the geodesic height of the successor and the geodesic height of v, then v can never be flooded in any feasible solution.

In order to find such non-flooded nodes, we start by computing the maximum possible geodesic height of each node than can be obtained after taking actions,[11] and then construct a new graph $G_{\text{nf}} = (V_{\text{nf}}, R_{\text{nf}})$[12] where each node is assigned its corresponding maximum possible geodesic height and where arcs are directed in downhill direction with respect to these geodesic heights. For each node on which no actions are located, we compute the set of its successors in G_{nf}.[13]

If the amount of rain that is needed to raise the water level at each of these successors to the absolute difference of the geodesic height of the successor and the geodesic height of v exceeds the total rain volume on the whole area, node v can never be flooded in any feasible solution. If this is not the case, we can apply the same idea using a larger set of nodes instead of the successors of v. To this end, we consider the undirected version of G_{nf} and remove all nodes that have strictly larger geodesic height than v. We then compute all nodes different from v that are in the same connected component as v in the remaining undirected graph. It is clear that the set of these nodes is a superset of the set of successors of v in G_{nf} and we can apply the same reasoning as before to this larger set of nodes.

The pseudocode of the corresponding algorithm is presented as Algorithm 7. Note that one could of course use the larger set of nodes right away, but this would cause a non-negligible overhead in computational effort.

Algorithm 7: PRESOLVE-NON-FLOODED

```
1  Procedure presolveNonFloodedNodes($G$)
2      Compute the maximal geodesic height for each node $v \in V$ and obtain $V_{\text{nf}}$.
3      Construct the graph $G_{\text{nf}} = (V_{\text{nf}}, R_{\text{nf}})$ by adding downhill arcs
4      Initialize presolve_non_flooded = $\varnothing$
5      for $v \in V_{nf}$ do
6          volume_needed = 0
7          if $v \neq s$ and $\mathcal{A}(v) = \varnothing$ then
8              for $v' \in$ successors($v$) do
9                  volume_needed = volume_needed + area$_{v'} \cdot (\text{gh}(v) - \text{gh}v')$
10             if volume_needed > total rain volume then
11                 Add $v$ to presolve_non_flooded
12             else
13                 Take a copy of the undriected version of $G_{\text{nf}}$ and remove all nodes with
                     geodesic height larger than GH($v$)
14                 volume_needed = 0
15                 for each $v'$ in the connected component of v after removal with $v' \neq v$ do
16                     volume_needed = volume_needed + area$_{v'} \cdot (\text{gh}(v) - \text{gh}v')$
17                 if volume_needed > total rain volume then
18                     Add $v$ to presolve_non_flooded
19     return presolve_non_flooded
```

[11]Recall that building embankments can increase the geodesic heights of nodes.

[12]**nf**: **n**on-**f**looded

[13]Note that nodes on which no actions are located have the same geodesic height in G and in G_{nf}.

3.4 Validity of the Mixed-Integer-Programming Formulation

In this section, we prove that the MIP is a valid formulation of the problem described in Section 3.3.1. To formally define the statement we want to prove, we call a subset $D \subseteq \mathcal{A}$ of actions such that building exactly the actions in D fulfills the budget Constraint (40) and does not violate any bounds on the incentives in Constraints (41)–(44) a *feasible set of actions.* The section is subdivided into two parts. In the first part, we show that there exists a solution x of the MIP for each feasible set D of actions. In the second part, we show that each solution x of the MIP that takes exactly the actions in a feasible set D of actions yields the same water levels as the result y of Algorithm 3 applied on G^D. In particular, the statement of the second part implies that the objective value of a solution of the MIP only depends on the taken actions. While the two proofs follow a similar idea, the proof in the first part is significantly shorter and provides an intuition for the proof in the second part.

3.4.1 Assumptions and Structural Results

Due to the similarity of the two proofs, we first present the required assumptions and some structural results that apply for both proofs.

For both proofs, we make the following assumptions:

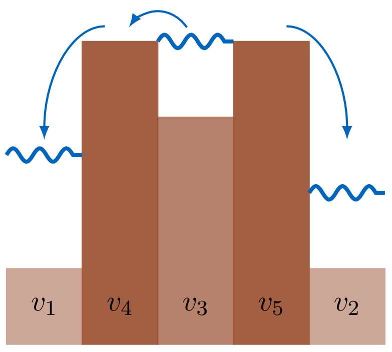
(A1) The graph G and, hence, also the graph G^D are weakly connected. This is the case in all realistic instances and, moreover, can be assumed without loss of generality because the arguments in the proofs can be applied to each weakly connected component individually in case that there are multiple weakly connected components.

(A2) The highest node in the graph G is non-flooded in any solution x of the MIP and any result y from Algorithm 3 with input graph G^D for a feasible set D of actions. This assumption is satisfied in all real-world problem instances since rain events that flood each single node are unrealistic, and damage on buildings could not be mitigated by any realistic actions anyway in such cases.

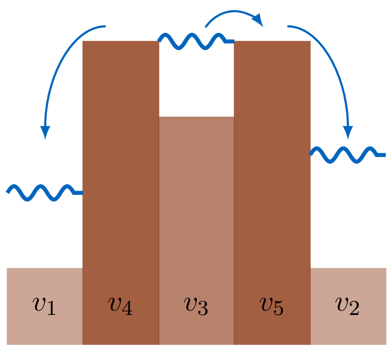
(A3) The geodesic heights of nodes in G^D are pairwise distinct, which is true if $G = G_{\text{red}}$ and $D = \varnothing$. In the case that $G = G_{\text{red}}$ and $D \neq \varnothing$, pairwise distinct geodesic heights of the nodes in G^D can be enforced by adding a slight noise to the heights or depths of the actions in a similar manner as in Algorithm 6.

It is important to note that Assumption (A3) is indeed required to ensure that, given a set of taken actions, the water levels in all solutions taking these actions are unique. This is corroborated in the following example.

Example 3.1 Let $G = (V, R)$ with $V = \{v_1, \ldots, v_5\}$ with $\text{gh}(v_1) = \text{gh}(v_2) = 0\text{m}$, $\text{gh}(v_3) = 9\text{m}$, and $\text{gh}(v_4) = \text{gh}(v_5) = 10\text{m}$. Each node has an area of 1sqm. Node v_5 is adjacent to v_2 and v_3 and node v_4 is adjacent to v_1 and v_3. The set $\mathcal{A}$ of possible actions is empty and the total rain per sqm is 2m. Since the water level at v_3 is 1m in each feasible solution, its remaining initial water of 1m^3 can be arbitrarily distributed among its incident arcs.



(a) In solution $x^{(1)}$, the remaining intitial water from v_3 is sent to v_4 and from there to v_1, which yields $\text{wl}(v_1) = 5\text{m}$ and $\text{wl}(v_2) = 4\text{m}$.



(b) In solution $x^{(2)}$, the remaining intitial water from v_3 is sent to v_5 and from there to v_2, which yields $\text{wl}(v_1) = 4\text{m}$ and $\text{wl}(v_2) = 5\text{m}$.

Figure 3.5.: An illustration of the two solutions in Example 3.1

By sending the whole water to v_4 or v_5, we obtain two solutions $x^{(1)}$ and $x^{(2)}$ which have different water levels in the nodes v_1 and v_2. The two solutions are illustrated in Figure 3.5.

As some variables are denoted the same in the MIP and in Algorithm 3, for a solution x of the MIP and the result y of Algorithm 3, we write, e.g., $x.\text{wl}$ and $y.\text{wl}$, respectively, in case of the water levels, to distinguish between them whenever the distinction is not clear from the context.

Next, we argue that it suffices for both proofs to show the statement for $D = \varnothing$, which is clearly a set of feasible actions. To this end, let D be a feasible set of actions and let x be a solution that takes exactly the actions in D . We observe that Constraints (3)–(14) imply that, for each $v \in V$, it holds that $x.\text{gh}(v)$ is the geodesic height of v in G^D. We therefore omit the "x." for the geodesic height in the following. Also note that taking actions only directly affects the geodesic heights in x, but the flows and, hence, the water levels are only affected indirectly from taking actions via their dependence on the geodesic heights in G^D. Further, the variables $\text{od}(r)$ for $r \in R$ only act as a case distinction to activate or deactivate constraints in the MIP. Hence, a solution x_D of the MIP with input graph G^D where no actions can be taken can immediately be constructed from x and vice versa. Since the two solutions yield the same water levels, it suffices to show the statements for the case where $D = \varnothing$ and, thus, $G^D = G$, i.e., for the case where no actions are taken.

We continue by presenting definitions that are required for both proofs and point out structural results that are mainly shown during the second proof in Section 3.4.3. To this end, in the following, we let x be a solution of the MIP taking no actions at all and let y be the result of Algorithm 3 with input graph G.

Inclusionwise-maximum weakly connected subgraphs of flooded nodes play an important role in both proofs and, hence, deserve an own definition.

Definition 3.2 Given x or y, each weakly connected component of the subgraph of G that is induced by the set of flooded nodes is called a *sink*. The set of all sinks is denoted by $\mathcal{S}(x)$ and $\mathcal{S}(y)$ for x and y, respectively.

If a node is flooded, then all its successors must be flooded as well, which immediately imposes a certain structure of sinks. We introduce the even stronger notion of *pre-sinks*. This idea has already been used in Algorithm 7 in line 13 without formally introducing the notion when presetting nodes to be non-flooded.

Definition 3.3 For $v \in V$, the weakly connected component containing v in the induced subgraph $G_{\leqslant v} := G|_{\{v' \in V : \mathrm{gh}(v') \leqslant \mathrm{gh}(v)\}}$ is called the *pre-sink induced by* v and is denoted by $\mathrm{PS}(v)$.

In the following, we slightly abuse notation by identifying a sink or pre-sink with the set of its nodes as long as this does not lead to any confusion.

It is shown in Section 3.4.3.1 that, for both x and y, each sink is a pre-sink, which motivates the investigation of the structure of pre-sinks in further detail. One important property is that two pre-sinks are either disjoint or one of them is a subgraph of the other, which is formally shown in the following lemma.

Lemma 3.4 Let $u, v \in V$ with $\mathrm{gh}(u) > \mathrm{gh}(v)$. Then it either holds that $\mathrm{PS}(v) \cap \mathrm{PS}(u) = \varnothing$ or it holds that $\mathrm{PS}(v) \subsetneq \mathrm{PS}(u)$.

Proof. Let C be the set of nodes of the weakly connected component in $G_{\leqslant u}$ that contains v. Then it immediately follows that $\mathrm{PS}(v) \subseteq C$. If it holds that $\mathrm{PS}(v) \cap \mathrm{PS}(u) \neq \varnothing$, it also holds that $C \cap \mathrm{PS}(u) \neq \varnothing$. It follows that $\mathrm{PS}(v) \subseteq C = \mathrm{PS}(u)$. Since $\mathrm{gh}(u) > \mathrm{gh}(v)$, it holds that $u \notin \mathrm{PS}(v)$, which completes the proof. $\square$

As a side remark, the statement of Lemma 3.4 induces that the set $\{\mathrm{PS}(u) \mid u \in V\}$ is laminar. Another useful result is that, when removing the highest node and all its incident arcs in a pre-sink, each weakly connected component of the obtained graph is again a pre-sink. This is shown in the following lemma and definition.

Lemma and Definition 3.5 Let $v \in V$. Then all weakly connected components of $G|_{\mathrm{PS}(v) \setminus \{v\}}$ are pre-sinks. We call a node u inducing such a pre-sink a *follow-up node* of v and denote the set of all follow-up nodes of v by $\mathrm{FUN}(v)$.

Proof. Let C be a weakly connected component of $G|_{\mathrm{PS}(v) \setminus \{v\}}$ and let u be the highest node in C. We show that $C = \mathrm{PS}(u)$.

Let $w \in C$. As C is weakly connected and u is the highest node in C, there exists an undirected path with trace $(u, \tilde{u}_1, \ldots, \tilde{u}_k, w)$ where all intermediate nodes are in C as well. This means that $w \in \mathrm{PS}(u)$.

Let $w \in \mathrm{PS}(u)$. This means there exists an undirected path P with trace $(u, \tilde{u}_1, \ldots, \tilde{u}_k, w)$ of nodes in $\mathrm{PS}(u)$. As $u \in \mathrm{PS}(v)$, it holds that $\mathrm{PS}(u) \subsetneq \mathrm{PS}(v)$ due to Lemma 3.4, which implies that $\tilde{u}_i \in \mathrm{PS}(v)$ for all $i \in \{1, \ldots, k\}$. Further, as $\tilde{u}_i \in \mathrm{PS}(u)$, it holds that $\tilde{u}_i \neq v$ for all $i \in \{1, \ldots, k\}$. This implies P is an undirected path in $G|_{\mathrm{PS}(v) \setminus \{v\}}$, which means that $w \in C$. $\square$

A central idea in the proofs of Sections 3.4.2 and 3.4.3 is to assess whether a pre-sink is indeed a sink. To this end, we need a measure for the volume of water a pre-sink can store, which is given in the following definition.

Definition 3.6 Let $v \in V$. If v is not the highest node in G, the *lowest parent* of the pre-sink $\mathrm{PS}(v)$, which is denoted by $\mathrm{lp}(\mathrm{PS}(v))$, is defined as the node with minimum geodesic height in $\delta_G^-(\mathrm{PS}(v))$.[14] If v is the highest node in G, we set $\mathrm{lp}(\mathrm{PS}(v)) := v$ to avoid notation issues. Further, the *threshold* of the pre-sink $\mathrm{PS}(v)$ is defined as

$$\mathrm{thr}(\mathrm{PS}(v)) := \sum_{u \in \mathrm{PS}(v)} (\mathrm{gh}(v) - \mathrm{gh}(u)) \cdot \mathrm{area}(u)$$

and the *capacity* of the pre-sink $\mathrm{PS}(v)$ is defined as

$$\mathrm{cap}(\mathrm{PS}(v)) := \sum_{u \in \mathrm{PS}(v)} (\mathrm{gh}(\mathrm{lp}(\mathrm{PS}(v))) - \mathrm{gh}(u)) \cdot \mathrm{area}(u).$$

As an intuition, the threshold of a pre-sink is the maximum amount of water the pre-sink can hold before all nodes of the pre-sink become flooded, and the capacity is the maximum amount of water the pre-sink can hold before its water level has reached the geodesic height of its lowest parent. These values become particularly important in a later stage of the proof where they are used to characterize whether the highest node in a pre-sink is flooded and how the flows between the pre-sink and its lowest parent behave.

The construction of the nodes in $\mathrm{FUN}(v)$ for some $v \in V$ in the proof of Lemma and Definition 3.5 and the definition of the lowest parent of a pre-sink immediately yield an important property of v and its follow-up nodes.

Corollary 3.7 Let $v \in V$ and $u \in \mathrm{FUN}(v)$, then $v = \mathrm{lp}(\mathrm{PS}(u))$.

It is shown in Section 3.4.3.2 that all nodes in a pre-sink are flooded if and only if the amount of rain on the pre-sink plus the inflow from uphill nodes, which is exactly the positive contribution to the excess of the node, exceeds the pre-sink's threshold. This motivates the following definition.

Definition 3.8 Let $v \in V$. The *positive contribution to the excess* of the pre-sink $\mathrm{PS}(v)$ in x is defined as

$$x.\mathrm{pce}(\mathrm{PS}(v)) := \sum_{r \in \delta_{G^{\mathrm{ex}}}^-(\mathrm{PS}(v))} x.f(r) + \sum_{u \in \mathrm{PS}(v)} \mathrm{rain} \cdot \mathrm{area}(u).$$

If a pre-sink is indeed a sink, the flows on arcs between the pre-sink and its lowest parent play an important role, which is why we split the positive contribution to the excess into two parts.

Definition 3.9 Let $v \in V$. We define the *positive contribution to the excess from non-lowest parents* of pre-sink $\mathrm{PS}(v)$ as

[14]Note that the lowest parent exists in this case as G is assumed to be weakly connected. In general, it does not hold that $\mathrm{lp}(\mathrm{PS}(v)) = \mathrm{lp}(v)$.

$$x.\mathrm{pcenlp}(\mathrm{PS}(v)) := \sum_{\substack{r \in \delta^-_{G^{\mathrm{ex}}}(\mathrm{PS}(v)):\\ \alpha(r) \neq \mathrm{lp}(\mathrm{PS}(v))}} x.f(r) + \sum_{u \in \mathrm{PS}(v)} \mathrm{rain} \cdot \mathrm{area}(u)$$

and the *positive contribution to the excess from the lowest parent* of pre-sink $\mathrm{PS}(v)$ as

$$x.\mathrm{pcelp}(\mathrm{PS}(v)) := \sum_{\substack{r \in \delta^-_{G^{\mathrm{ex}}}(\mathrm{PS}(v)):\\ \alpha(r) = \mathrm{lp}(\mathrm{PS}(v))}} x.f(r).$$

Similar definitions for y, which we do not need for the proof of the existence of x, are provided in Sections 3.4.3.2 and 3.4.3.3.

It is shown in Section 3.4.3.3 that the behaviour of the flows between a pre-sink $\mathrm{PS}(v)$ and its lowest parent depend on whether $x.\mathrm{pcenlp}(\mathrm{PS}(v))$ exceeds the capacity of the pre-sink or not.

Before we start with the proof of the existence of x, we introduce one more definition, which simplifies notation when distributing water from a node to the pre-sinks induced by its follow-up nodes.

Definition 3.10 Let $v \in V$ and let $u \in \mathrm{FUN}(v)$. The *total ratio of the lowest parent* is defined by

$$\mathrm{ratiolp}(\mathrm{PS}(u)) := \sum_{\substack{r \in \delta^-_{G}(\mathrm{PS}(u)):\\ \alpha(r) = v}} \mathrm{ratio}(r).$$

3.4.2 Existence of x

We start by showing that, for each feasible set D of actions, there exists a solution x of the MIP that takes exactly the actions in D. As argued previously, it suffices to show the claim for $D = \varnothing$.

The proof is constructive and the idea is to firstly construct the water levels and the flows on the arcs that are incident to a non-flooded node and then to construct the flows on the other arcs. All other values of the non-trivial variables follow immediately from the flows and the water levels. Whenever we construct a positive flow on any arc $r \in R^{\mathrm{ex}}$, we also set the flow on its reversed arc $\overleftarrow{r}$ to zero. This ensures that the constructed solution fulfills Constraint (27).

We start by constructing the water levels and the flows on the arcs that are incident to a non-flooded node. Initially, we mark all nodes as unprocessed and process the nodes in G in order of decreasing geodesic height, hence, starting with the highest node $v' \in V$ in the graph. During the process, it is always ensured that the currently processed node v' is non-flooded. For the highest node in the graph, this is ensured by Assumption (A2). We, hence, set $x.\mathrm{wl}(v') := 0$. Next, note that, for any node $u \in \mathrm{FUN}(v')$, it is always ensured that the flows on arcs that contribute to $x.\mathrm{pcenlp}(\mathrm{PS}(u))$ have already been constructed since we process the nodes in a top-down manner. If it holds that $x.\text{pcenlp}(\text{PS}(u)) > \text{cap}(\text{PS}(u))$, we choose one arc $r \in \delta^+_{G^\text{ex}}(\text{PS}(u)) \cap \delta^-_{G^\text{ex}}(v')$ and set the flow on this arc to $x.f(r) := x.\text{pcenlp}(\text{PS}(u)) - \text{cap}(\text{PS}(u)) =: \text{bf}(\text{PS}(u))$.[15] The flows on all other arcs between v' and nodes in $\text{PS}(u)$ in G^ex are set to zero.

After this is done for all follow-up nodes of v', we proceed by distributing the water among the other outgoing arcs of v'. To this end, we define $\text{bfFUN}(v')$ as the set consisting of the follow-up nodes $u \in \text{FUN}(v')$ for which it holds that $x.\text{pcenlp}(\text{PS}(u)) > \text{cap}(\text{PS}(u))$ and the set $\text{nbfFUN}(v') := \text{FUN}(v') \backslash \text{bfFUN}(v')$. We now choose $\text{nfad}(v')$[16] such that:

$$\begin{aligned}
&\sum_{u \in \text{nbfFUN}(v')} \min\left\{\text{nfad}(v') \cdot \text{ratiolp}(\text{PS}(u)), \text{cap}(\text{PS}(u)) - \text{pcenlp}(\text{PS}(u))\right\} \\
= &\sum_{r \in \delta^-_G(v')} x.f(r) + \text{rain} \cdot \text{area}(v') + \sum_{u \in \text{bfFUN}(v')} \text{bf}(\text{PS}(u))
\end{aligned}$$

Note that such a value for $\text{nfad}(v')$ exists since the term on the left-hand side is continuous and monotonically increasing in $\text{nfad}(v')$ and it is ensured during the algorithm that $x.\text{pce}(\text{PS}(v')) \leqslant \text{thr}(\text{PS}(v'))$. In the first iteration, this again holds due to Assumption (A2). It is further worth noting that, if the minimum attains the value $\text{cap}(\text{PS}(u)) - \text{pcenlp}(\text{PS}(u))$ for some follow-up node $u \in \text{FUN}(v')$, then the amount of water in the pre-sink $\text{PS}(u)$ is exactly the pre-sink's capacity. This immediately yields that $\text{PS}(u)$ is a sink and that all arcs from v' to nodes in $\text{PS}(u)$ are full. In the other case, that is if the minimum attains a value strictly smaller than $\text{cap}(\text{PS}(u)) - \text{pcenlp}(\text{PS}(u))$, all arcs from v' to nodes in $\text{PS}(u)$ are non-full.

For each node $u \in \text{nbfFUN}(v')$, we distribute $\min\{\text{nfad}(v') \cdot \text{ratiolp}(\text{PS}(u), \text{cap}(\text{PS}(u)) - \text{pcenlp}(\text{PS}(u))\}$ units of flow along the arcs from v' to nodes in $\text{PS}(u)$ proportional to their ratios.

Note that, for each $u \in \text{FUN}(v')$, the flows on the arcs contributing to $x.\text{pce}(\text{PS}(u))$ have now been constructed. If it holds that $x.\text{pce}(\text{PS}(u)) > \text{thr}(\text{PS}(u))$, we set

$$x.\text{wl}(u) := \frac{\sum\limits_{v \in \text{PS}(u)} \text{rain} \cdot \text{area}(v) + \sum\limits_{r \in \delta^-_{G^\text{ex}}(\text{PS}(u))} x.f(r) - \sum\limits_{r \in \delta^+_{G^\text{ex}}(\text{PS}(u))} x.f(r) - \text{thr}(\text{PS}(u))}{\sum\limits_{v \in \text{PS}(u)} \text{area}(v)}$$

and for each $w \in \text{PS}(u) \backslash \{u\}$, we then set $x.\text{wl}(w) := x.\text{wl}(u) + \text{gh}(u) - \text{gh}(w)$. Further, we mark all nodes in $\text{PS}(u)$ as processed in this case.

The described method is then continued by processing the (unique) unprocessed node with largest geodesic height, until all nodes are processed.

Since, after all nodes in the graph have been processed, the water levels are constructed for all nodes in V, we set the excess at each node $v \in V$ to $x.\text{excess}(v) := x.\text{wl}(v) \cdot \text{area}(v)$. We complete the construction by presenting a method to construct the flows on arcs between two flooded nodes. To this end, we introduce the feasible flow problem (FFP):

[15]The notation "bf" stands for **b**ack**f**loat and will be introduced in more detail in Section 3.4.3.3.

[16]The notation "nfad" stands for **n**on-**f**ull **a**rc **d**istribution.

Feasible Flow Problem (FFP)

INSTANCE:	A directed graph $H = (V_H, R_H)$ and demands $b : V_H \to \mathbb{Q}$ with $\sum_{v \in V_H} b(v) = 0$
TASK:	Find a flow $f \colon R_H \to \mathbb{Q}_{\geqslant 0}$ that satisfies the demand.

As pointed out in [AMO93], this problem can be transformed into a maximum flow problem and, therefore, can be solved efficiently. Also note that, due to the structure of G^{ex}, there exists a solution of FFP for any subgraph of G^{ex} that is induced by a connected set of nodes. Further, each such solution can be transformed such that, for any arc $r \in R$ that also is in the input graph of FFP, it holds that the flow on r or its reversed arc $\overleftarrow{r}$ is zero. In the following, when we speak about a solution of FFP, we implicitly assume that the flows in the solution fulfill this property.

Let $S \in \mathcal{S}(x)$ be a sink and let $u \in V$ such that $S = \text{PS}(u)$. The existence of such a node u follows immediately from the construction of the water levels. The flows on arcs in the sink are constructed by solving an instance of FFP where the input graph is the subgraph of G^{ex} induced by the set of nodes in the sink and the demand function is given by

$$b(v) := x.\text{excess}(v) - \text{rain} \cdot \text{area}(v) - \sum_{\substack{r \in \delta^-_{G^{\text{ex}}}(S): \\ \omega(r) = v}} x.f(r) + \sum_{\substack{r \in \delta^+_{G^{\text{ex}}}(S): \\ \alpha(r) = v}} x.f(r)$$

for each $v \in S$. We show that these demands fulfill $\sum_{v \in S} b(v) = 0$:

$$\begin{aligned}
\sum_{v \in S} b(v) &= \sum_{v \in S} \left[x.\text{excess}(v) - \text{rain} \cdot \text{area}(v) - \sum_{\substack{r \in \delta^-_{G^{\text{ex}}}(S): \\ \omega(r) = v}} x.f(r) + \sum_{\substack{r \in \delta^+_{G^{\text{ex}}}(S): \\ \alpha(r) = v}} x.f(r) \right] \\
&= \sum_{v \in S} [x.\text{excess}(v)] - \sum_{v \in S} \text{rain} \cdot \text{area}(v) - \sum_{r \in \delta^-_{G^{\text{ex}}}(S)} x.f(r) + \sum_{r \in \delta^+_{G^{\text{ex}}}(S)} x.f(r) \\
&= \sum_{v \in S} [x.\text{wl}(v) \cdot \text{area}(v)] - \sum_{v \in S} \text{rain} \cdot \text{area}(v) - \sum_{r \in \delta^-_{G^{\text{ex}}}(S)} x.f(r) + \sum_{r \in \delta^+_{G^{\text{ex}}}(S)} x.f(r) \\
&= \sum_{v \in S} [(x.\text{wl}(u) + \text{gh}(u) - \text{gh}(v)) \cdot \text{area}(v)] \\
&\quad - \sum_{v \in S} \text{rain} \cdot \text{area}(v) - \sum_{r \in \delta^-_{G^{\text{ex}}}(S)} x.f(r) + \sum_{r \in \delta^+_{G^{\text{ex}}}(S)} x.f(r) \\
&= \sum_{v \in S} [x.\text{wl}(u) \cdot \text{area}(v)] + \text{thr}(S) \\
&\quad - \sum_{v \in S} \text{rain} \cdot \text{area}(v) - \sum_{r \in \delta^-_{G^{\text{ex}}}(S)} x.f(r) + \sum_{r \in \delta^+_{G^{\text{ex}}}(S)} x.f(r)
\end{aligned}$$

Plugging in the constructed water level $x.\mathrm{wl}(u)$ yields that $\sum_{v \in S} b(v) = 0$. Hence, there exists a solution of the constructed instance of FFP, whose resulting flows on the subgraph are used to complete the construction of the flows in x.

We refrain from presenting the construction of the other variables of x since this is straightforward. It remains to show that the constructed solution x is indeed feasible. To this end, we only present the proofs for the fulfillment of Constraints (1) and (31) since the other constraints are clearly fulfilled by x.

To show the fulfillment of Constraint (1), let $v \in V$. If v is in a sink, the constraint is fulfilled by the design of the demands in the instance of FFP that has been solved for the sink. If v is not in a sink, i.e., it is non-flooded, the whole amount of water that contributes with a positive sign to $x.\mathrm{excess}(v)$ in the constraint is distributed among the outgoing arcs of v by the choice of $\mathrm{nfad}(v)$ in the construction, which implies that Constraint (1) is also fulfilled in this case.

To show the fulfillment of Constraint (31), let $v \in V$ and $r_1, r_2 \in \delta^+_{G^{\mathrm{ex}}}(v)$. Recall that an arc is active if there is a positive flow on the arc. If v is flooded, by construction of the solution, the arcs r_1 and r_2 are each either full or not active. Hence, the constraint is not active in this case and therefore fulfilled automatically. If v is non-flooded, we have to distinguish three cases.

<u>Case 1:</u> Both arcs r_1 and r_2 are non-full and active.

Since both arcs are non-full and active and since v is non-flooded, both arcs must be downhill. Let $u^{(1)}$ be the unique node in $\mathrm{FUN}(v)$ such that $\omega(r_1) \in \mathrm{PS}(u^{(1)})$ and $u^{(2)}$ be the unique node in $\mathrm{FUN}(v)$ such that $\omega(r_2) \in \mathrm{PS}(u^{(2)})$. Since both arcs are non-full, it holds that $\mathrm{nfad}(v) \cdot \mathrm{ratiolp}(\mathrm{PS}(u^{(i)})) < \mathrm{cap}(\mathrm{PS}(u)) - \mathrm{pcenlp}(\mathrm{PS}(u))$ for each $i \in \{1, 2\}$. Hence, for $i \in \{1, 2\}$, the flow on r_i is

$$x.f(r_i) = \mathrm{nfad}(v) \cdot \mathrm{ratiolp}(\mathrm{PS}(u^{(i)})) \cdot {}^{\mathrm{ratio}(r_i)}\!/_{\mathrm{ratiolp}(\mathrm{PS}(u^{(i)}))} = \mathrm{nfad}(v) \cdot \mathrm{ratio}(r_i).$$

This proves that the flows are distributed proportionally on the arcs r_1 and r_2 according to the ratios and that Constraint (31) is fulfilled.

<u>Case 2:</u> Only one of the arcs is non-full and active.

Without loss of generality, let r_1 be non-full and active. In this case, it remains to show that $x.f(r_2) \leqslant {}^{\mathrm{ratio}(r_2)}\!/_{\mathrm{ratio}(r_1)} \cdot x.f(r_1)$. If r_2 is not active, it holds that $f(r_2) = 0$ and, hence, the required inequality holds.

If r_2 is active, it must be full. As before, let $u^{(2)}$ be the unique node in $\mathrm{FUN}(v)$ such that $\omega(r_2) \in \mathrm{PS}(u^{(2)})$. Since r_2 is full, it holds that $\mathrm{nfad}(v) \cdot \mathrm{ratiolp}(\mathrm{PS}(u^{(i)})) \geqslant \mathrm{cap}(\mathrm{PS}(u)) - \mathrm{pcenlp}(\mathrm{PS}(u))$ and the flow on r_2 is

$$\begin{aligned} x.f(r_2) &= (\mathrm{cap}(\mathrm{PS}(u)) - \mathrm{pcenlp}(\mathrm{PS}(u))) \cdot \frac{\mathrm{ratio}(r_2)}{\mathrm{ratiolp}(\mathrm{PS}(u^{(2)}))} \\ &\leqslant \mathrm{nfad}(v) \cdot \mathrm{ratiolp}(\mathrm{PS}(u^{(2)})) \cdot \frac{\mathrm{ratio}(r_2)}{\mathrm{ratiolp}(\mathrm{PS}(u^{(2)}))} \\ &= \mathrm{nfad}(v) \cdot \mathrm{ratio}(r_2). \end{aligned}$$

By the same argument as in the first case for r_1, the desired inequality is shown.

Case 3: None of the arcs is non-full and active.

In this case, there is nothing to show since Constraint (31) is not active and, hence, fulfilled automatically.

All in all, this yields the desired theorem.

Theorem 3.12 Let $D \subseteq \mathcal{A}$ be a feasible set of actions, then there exists a feasible solution of the MIP that takes exactly the actions in D.

3.4.3 Equality of Water Levels

The aim of the second proof is to show that each solution x of the MIP that takes exactly the actions in a feasible set D of actions yields the same water levels as the result y of Algorithm 3 applied on G^D. As the value of the objective is determined by the water levels, this in particular means that the objective value of a solution of the MIP only depends on the taken actions.

As argued in Section 3.4.1, it suffices to show the claim for $D = \emptyset$. Hence, in the following, we let x be an arbitrary but fixed feasible solution of the MIP that takes no actions at all and we let y be the result of Algorithm 3 with input graph G. The existence of x has previously been shown in Section 3.4.2.

It is worth noting that, although we prove that the water levels in any feasible solution x of the MIP taking a given set D of actions coincide with those of the corresponding result y of Algorithm 3 applied on G^D, the flows in different solutions of the MIP taking the same actions can still differ. The reason is that, if a cycle of flooded nodes of length at least three exists in G^{ex}, an arbitrary amount of flow might be sent over this cycle, which conserves feasibility but causes the flows to be different in the two obtained solutions even when the same actions are taken.

3.4.3.1 Characterization of Flooded Subgraphs

As already announced in Section 3.4.1, in this section, we prove that, for both x and y, every sink is a pre-sink. To this end, the reader is advised to revisit Definitions 3.2 and 3.3.

Characterization of Flooded Subgraphs for x

We start by proving the desired connection between sinks and pre-sinks for the feasible solution x of the MIP. Throughout this section, we omit the “$x.$” when referring to variables, so, e.g., the water level at a node v is denoted by $\text{wl}(v)$ instead of $x.\text{wl}(v)$.

First, two properties of the water levels of adjacent nodes are observed.

Observation 3.13 Let $r \in R$ with $\alpha(r) = u$, $\omega(r) = v$. If $\text{wl}(u) > 0$, then $\text{wl}(v) + \text{gh}(v) = \text{wl}(u) + \text{gh}(u)$.

Proof. Constraints (46.1) and (46.2) imply that r is full. Constraint (36) then yields $\text{wl}(v) + \text{gh}(v) = \text{wl}(u) + \text{gh}(u)$. □

Observation 3.14 Let $r \in R$ with $\alpha(r) = u$, $\omega(r) = v$. If $\mathrm{wl}(v) > \mathrm{gh}(u) - \mathrm{gh}(v)$, then $\mathrm{wl}(v) + \mathrm{gh}(v) = \mathrm{wl}(u) + \mathrm{gh}(u)$.

Proof. Constraint (22) forces $\mathrm{full}(r) = 1$ as $\mathrm{wl}(v) > \mathrm{gh}(u) - \mathrm{gh}(v)$. As before, Constraint (36) then yields $\mathrm{wl}(v) + \mathrm{gh}(v) = \mathrm{wl}(u) + \mathrm{gh}(u)$. $\square$

The two observations are used to prove the following important proposition:

Proposition 3.15 Let $v \in V$. If $\mathrm{gh}(v') + \mathrm{wl}(v') > \mathrm{gh}(v)$ for some $v' \in \mathrm{PS}(v)$, then every node $\hat{v} \in \mathrm{PS}(v)$ is flooded with $\mathrm{gh}(\hat{v}) + \mathrm{wl}(\hat{v}) = \mathrm{gh}(v') + \mathrm{wl}(v')$.

Proof. Let $v' \in \mathrm{PS}(v)$ with $\mathrm{gh}(v') + \mathrm{wl}(v') > \mathrm{gh}(v)$ and let $\hat{v}$ be an arbitrary node in $\mathrm{PS}(v)$. Then, since the pre-sink is weakly connected, there exists an undirected path P with $\mathrm{trace}(P) = (v', \tilde{v}_1, \dots, \tilde{v}_k, \hat{v})$ in $G_{\leqslant v}$. In particular, it holds that $\mathrm{gh}(\tilde{v}_i) \leqslant \mathrm{gh}(v)$ for all $i \in \{1, \dots, k\}$. Applying Observations 3.13 and 3.14 inductively on the path yields

$$\begin{aligned} \mathrm{wl}(\tilde{v}_i) &= \mathrm{wl}(v') + \mathrm{gh}(v') - \mathrm{gh}(\tilde{v}_i) > 0 \text{ for all } i \in \{1, \dots, k\}, \text{ and} \\ \mathrm{wl}(\hat{v}) &= \mathrm{wl}(v') + \mathrm{gh}(v') - \mathrm{gh}(\hat{v}) > 0, \end{aligned}$$

which proves the claim. $\square$

Using Proposition 3.15, we prove the desired connection between sinks and pre-sinks for x:

Proposition 3.16 Let $S \in \mathcal{S}(x)$ and let $v \in S$ be the node with largest geodesic height among all nodes in S. Then $S = \mathrm{PS}(v)$.

Proof. By definition of $\mathrm{PS}(v)$, it holds that $v \in \mathrm{PS}(v)$. Moreover, it is easy to see that $\mathrm{PS}(v) \subseteq S$ by applying Proposition 3.15 for $v' = v$. Hence, it only remains to show that $S \subseteq \mathrm{PS}(v)$. To this end, let $v' \in S$ be an arbitrary node. By the definition of a sink, S is weakly connected, which means that there exists an undirected path P in G only containing nodes in S with $\mathrm{trace}(P) = (v, \tilde{v}_1, \dots, \tilde{v}_k, v')$ and $\mathrm{gh}(\tilde{v}_i) \leqslant \mathrm{gh}(v)$ for all $i \in \{1, \dots, k\}$. This means that P is also an undirected path in $G_{\leqslant v}$ and, hence, that v' is in the same connected component of $G_{\leqslant v}$ as v. Therefore, we obtain that $v' \in \mathrm{PS}(v)$. $\square$

The proofs of Propositions 3.15 and 3.16 also yield the following helpful property about the water levels at nodes within the same sink:

Corollary 3.17 Let $S \in \mathcal{S}(x)$ and $u, v \in S$. Then $\mathrm{wl}(v) + \mathrm{gh}(v) = \mathrm{wl}(u) + \mathrm{gh}(u)$.

When investigating the water levels in x, it therefore suffices to know the water level at one node per sink.

Characterization of Flooded Subgraphs for y

We now prove the desired connection between sinks and pre-sinks for the result y of Algorithm 3. Throughout this section, we again omit the "y." when referring to variables, so, e.g., the water level at a node v is denoted by $\mathrm{wl}(v)$ instead of $y.\mathrm{wl}(v)$. Although the proof is a bit more involved, the basic idea is similar to the proof of Proposition 3.16. Before we show two observations similar to Observations 3.13 and 3.14, we introduce some notation and obtain further structural results.

Notation 3.18 The total number of iterations of the while-loop in Algorithm 3 is denoted by T. Further, we write $\overline{G} = (\overline{V}, \overline{R}) := (\tilde{V}_T, \tilde{R}_T)$ and $\text{repres}(v) := \text{repres}_T(v)$ for $v \in V$.

Further, let $t \in \{1, \dots, T-1\}$ and let $v \in V$ such that v is the first flooded leaf in iteration t. We then say that v *leaves the graph* in iteration t. Given a node $v \notin \overline{V}$, we denote the unique iteration in which v leaves the graph by t_v. In the following, we present some structural results about the sets $\text{repres}_t(v)$ for $t \in \{1, \dots, T\}$ and $v \in V$. To this end, we introduce a definition for the water levels after some iteration t.

Definition 3.19 Let $t \in \{1, \dots, T\}$ and $v \in V$. The *water level of v after iteration t* is defined as the water level that is obtained if the while loop in line 8 of Algorithm 3 is exited after the t-th iteration and is denoted by $\widetilde{\text{wl}}_t(v)$.

Note that it holds that $\widetilde{\text{wl}}_T(v) = \text{wl}(v)$ for all $v \in V$. We start with an observation of the water levels after an iteration t.

Observation 3.20 Let $t \in \{1, \dots, T\}$ and $v \in \tilde{V}_t$. For two nodes $u_1, u_2 \in \text{repres}_t(v)$, it holds that $\text{gh}(u_1) + \widetilde{\text{wl}}_t(u_1) = \text{gh}(u_2) + \widetilde{\text{wl}}_t(u_2)$

Proof. This follows immediately from lines 20 to 22 of the algorithm. □

As shown in the previous observation, the water level at a node $v \notin \overline{V}$ is determined by the water level of a node $\bar{v} \in \overline{V}$ with $v \in \text{repres}(\bar{v})$. We now introduce a suitable notion for this node and show that it is uniquely defined.

Lemma and Definition 3.21 For each $t \in \{1, \dots, T\}$ and each node $v \in V$, there exists exactly one node $\tilde{v} \in \tilde{V}_t$ such that $v \in \text{repres}_t(\tilde{v})$. For a node $v \in V$, we call the (unique) node $\bar{v} \in \overline{V}$ such that $v \in \text{repres}(\bar{v})$ the *highest representative of v* and write $\bar{v} = \text{hr}(v)$.

Proof. For $t = 1$ the claim is clear as $\text{repres}_1(v) = \{v\}$ for all $v \in V$. In each iteration $t \in \{1, \dots, T-1\}$ of the algorithm, one node u leaves the graph and is joined with its lowest parent v. All nodes in $\text{repres}_t(u)$ are then in $\text{repres}_{t+1}(v)$ in the next iteration and all other sets $\text{repres}_t(v')$ for $v' \in \tilde{V}_t \backslash \{v\}$ remain unchanged. Hence, the property is conserved in each iteration of the algorithm, which proves the claim. □

Note that the set $\text{repres}_{t_v}(v)$ is not deleted when a node $v \in V$ leaves the graph, which means a node $v \in V$ can be in several sets $\text{repres}(v')$ for $v' \in V$, but only for one of them, it holds that $v' \in \overline{V}$. Also note that, as soon as a node $v \in V$ joins a set $\text{repres}_t(u)$ for some other node $u \in V$ in some iteration $t \in \{1, \dots, T-1\}$, the node never leaves this set again, which implies that $v \in \text{repres}(u)$ in this case. Moreover, the previous proof yields the following observation:

Observation 3.22 Let $v \in V$. Then $v \in \overline{V}$ if and only if $v = \mathrm{hr}(v)$.

We proceed by proving some further structural results in order to obtain the analogous statements to Observations 3.13 and 3.14 in the context of y.

Lemma 3.23 Let $t \in \{1, \dots, T\}$ and $r \in R$ with $\alpha(r) = u$ and $\omega(r) = v$. If $u \notin \tilde{V}_t$, then it holds that $v \in \mathrm{repres}_t(u)$.

Proof. We start by showing that also $v \notin \tilde{V}_t$. Suppose for the sake of a contradiction that $v \in \tilde{V}_t$. Then the child v of u never leaves the graph until iteration t, which implies that u is never a leaf until iteration t. Consequently, u cannot be removed until iteration t and, thus, $u \in \tilde{V}_t$, which contradicts the assumption that $u \notin \tilde{V}_t$.

Since $v \notin \tilde{V}_t$, there exists an iteration $t_v < t$ where v leaves the graph, i.e., `joinNodes`$(v, \tilde{v}_1)$ is called for the lowest parent $\tilde{v}_1 := \mathrm{lp}_{\tilde{G}_{t_v}}(v)$ of v in the graph of iteration t_v. Thus, we have $v \in \mathrm{repres}_{t_v+1}(\tilde{v}_1)$ in the following iteration, which also implies that $v \in \mathrm{repres}_t(\tilde{v}_1)$.

If $\tilde{v}_1 = u$, we are done. Otherwise, since $\tilde{v}_1$ is the lowest parent of v in $\tilde{G}_{t_v}$, it holds that $\mathrm{gh}(u) > \mathrm{gh}(\tilde{v}_1)$ and there exists an arc $r_1 \in \tilde{R}_{t_v+1}$ from u to $\tilde{v}_1$. In the same way as for v, it then follows that $\tilde{v}_1 \notin \tilde{V}_t$ and that $\tilde{v}_1$ must, hence, be joined into another node $\tilde{v}_2 \in V$ in some iteration $t_{\tilde{v}_1} < t$, which implies that $v \in \mathrm{repres}_t(\tilde{v}_2)$. Applying this argument iteratively induces a sequence of nodes with strictly increasing geodesic heights until eventually $\tilde{v}_k = u$ for some $k \in \mathbb{N}$. Thus, it holds that $v \in \mathrm{repres}_t(u)$. □

The proof of the next lemma is highly similar to the proof of Lemma 3.23.

Lemma 3.24 Let $t \in \{1, \dots, T\}$ and $r \in R$ with $\alpha(r) = u$ and $\omega(r) = v$. If $\widetilde{\mathrm{wl}}_t(u) > 0$, then $v \in \mathrm{repres}_t(u)$.

Proof. If $u \notin \tilde{V}_t$, the claim follows directly from Lemma 3.23. Therefore, let $u \in \tilde{V}_t$. As it holds that $\widetilde{\mathrm{wl}}_t(u) > 0$, the node u must be a leaf in $\tilde{G}_t$, which implies that $v \notin \tilde{V}_t$. Hence, there exists an iteration $t_v < t$ where v leaves the graph, i.e., `joinNodes`$(v, \tilde{v}_1)$ is called for $\tilde{v}_1 := \mathrm{lp}_{G_{t_v}}(v)$. As in the proof of Lemma 3.23, this implies that $v \in \mathrm{repres}_t(\tilde{v}_1)$.

If $\tilde{v}_1 = u$, we are done. In the other case, we construct a sequence $(\tilde{v}_1, \dots, \tilde{v}_k = u)$ of nodes with strictly increasing geodesic heights analogously to the proof of Lemma 3.23, while $\tilde{v}_i \notin \tilde{V}_t$ for all $i \in \{1, \dots, k-1\}$ follows since, otherwise, u would not be a leaf in $\tilde{G}_t$. □

One further structural result is required, which can be interpreted as the transitivity of the representatives.

Observation 3.25 Let $u, v, w \in V$ such that $u \in \mathrm{repres}(v)$ and $v \in \mathrm{repres}(w)$. Then $u \in \mathrm{repres}(w)$.

Proof. Let u join $\mathrm{repres}(v)$ in iteration $\hat{t}_u \geqslant t_u$, and let v join $\mathrm{repres}(w)$ in iteration $\hat{t}_v \geqslant t_v$. Since $\mathrm{repres}(v)$ remains unchanged after v leaves the graph, it must hold that $\hat{t}_u < t_v \leqslant \hat{t}_v$. Moreover, in all iterations $t > \hat{t}_u$, the nodes u and v are always in the same set $\mathrm{repres}_t(\tilde{v})$ for $\tilde{v} \in \tilde{V}_t$. Thus, when v joins $\mathrm{repres}(w)$ in iteration $\hat{t}_v > \hat{t}_u$, so does u, which proves the claim. □

The technical results above allow proving the following observation, which is the analogue of Observation 3.13 in the context of y:

Observation 3.26 Let $r \in R$ with $\alpha(r) = u$ and $\omega(r) = v$. If $\mathrm{wl}(u) > 0$ then $\mathrm{hr}(u) = \mathrm{hr}(v)$ and $\mathrm{wl}(u) + \mathrm{gh}(u) = \mathrm{wl}(v) + \mathrm{gh}(v)$.

Proof. Lemma 3.24 applied for $t = T$ implies that $v \in \mathrm{repres}(u)$. From Observation 3.25, we obtain that $v \in \mathrm{repres}(\mathrm{hr}(u))$ and, hence, that $\mathrm{hr}(u) = \mathrm{hr}(v)$. Observation 3.20 applied for $t = T$ then implies that $\mathrm{wl}(u) + \mathrm{gh}(u) = \mathrm{wl}(v) + \mathrm{gh}(v)$. □

To prove the analogue of Observation 3.14, we need one more structural result:

Lemma 3.27 Let $t \in \{1, \dots, T\}$ and $r \in R$ with $\alpha(r) = u$ and $\omega(r) = v$. If $\widetilde{\mathrm{wl}}_t(v) > \mathrm{gh}(u) - \mathrm{gh}(v)$, then $v \in \mathrm{repres}_t(u)$.

Proof. We start by proving that $v \notin \tilde{V}_t$. For the sake of a contradiction, suppose that $v \in \tilde{V}_t$. Due to Lemma 3.23, it must then hold that $u \in \tilde{V}_t$ as, otherwise, this would imply that $v \in \mathrm{repres}_t(u)$ and, hence, that $v \notin \tilde{V}_t$. Therefore, the arc r is never removed or changed until iteration t, so $r \in \tilde{R}_{t'}$ for all $t' \in \{1, \dots, t\}$. As $\widetilde{\mathrm{wl}}_t(v) > \mathrm{gh}(u) - \mathrm{gh}(v)$ and $v \in \tilde{V}_t$, the node v must be a leaf in $\tilde{G}_t$. Let $\tilde{v} = \mathrm{lp}_{\tilde{G}_t}(v)$. It must hold that $\widetilde{\mathrm{wl}}_t(v) \leqslant \mathrm{gh}(\tilde{v}) - \mathrm{gh}(v)$ as, otherwise, v would have been joined into its lowest parent until iteration t. As u is a parent of v in each iteration $t' \leqslant t$, it must hold that $\mathrm{gh}(\tilde{v}) \leqslant \mathrm{gh}(u)$. This implies that $\widetilde{\mathrm{wl}}_t(v) \leqslant \mathrm{gh}(u) - \mathrm{gh}(v)$, which is a contradiction to $\widetilde{\mathrm{wl}}_t(v) > \mathrm{gh}(u) - \mathrm{gh}(v)$. Thus, it holds that $v \notin \tilde{V}_t$.

As $v \notin \tilde{V}_t$, there exists an iteration $t_v < t$ where v leaves the graph, i.e., `joinNodes`$(v, \tilde{v}_1)$ is called for $\tilde{v}_1 := \mathrm{lp}_{\tilde{G}_{t_v}}(v)$. Hence, it holds that $v \in \mathrm{repres}_t(\tilde{v}_1)$. If $\tilde{v}_1 = u$, we are done. Otherwise, we construct a sequence $(\tilde{v}_1, \dots, \tilde{v}_k = u)$ of nodes with strictly increasing geodesic heights analogously to the proof of Lemma 3.23, while $\tilde{v}_i \notin \tilde{V}_t$ for all $i \in \{1, \dots, k-1\}$ follows since, otherwise, it holds that $v \in \mathrm{repres}_t(\tilde{v}_i)$ and Observation 3.20 implies that $\widetilde{\mathrm{wl}}_t(\tilde{v}_i) = \widetilde{\mathrm{wl}}_t(v) + \mathrm{gh}(v) - \mathrm{gh}(\tilde{v}_i) > \mathrm{gh}(u) - \mathrm{gh}(\tilde{v}_i)$. The desired contradiction is then obtained analogously to the argumentation above. □

Observation 3.28 Let $r \in R$ with $\alpha(r) = u$ and $\omega(r) = v$. If $\mathrm{wl}(v) > \mathrm{gh}(u) - \mathrm{gh}(v)$, then it holds that $\mathrm{hr}(u) = \mathrm{hr}(v)$ and $\mathrm{wl}(u) + \mathrm{gh}(u) = \mathrm{wl}(v) + \mathrm{gh}(v)$.

Proof. Lemma 3.27 applied for $t = T$ implies that $v \in \mathrm{repres}(u)$. From Observation 3.25, we obtain that $v \in \mathrm{repres}(\mathrm{hr}(u))$ and, hence, that $\mathrm{hr}(u) = \mathrm{hr}(v)$. Observation 3.20 applied for $t = T$ then implies that $\mathrm{wl}(u) + \mathrm{gh}(u) = \mathrm{wl}(v) + \mathrm{gh}(v)$. □

We now use Observations 3.26 and 3.28 to prove the analogue of Proposition 3.15:

Proposition 3.29 Let $v \in V$. If $\mathrm{gh}(v') + \mathrm{wl}(v') > \mathrm{gh}(v)$ for some $v' \in \mathrm{PS}(v)$, then every node $\hat{v} \in \mathrm{PS}(v)$ is flooded with $\mathrm{gh}(\hat{v}) + \mathrm{wl}(\hat{v}) = \mathrm{gh}(v') + \mathrm{wl}(v')$.

Proof. The proof is completely analogous to the proof of Proposition 3.15 except for using Observations 3.26 and 3.28 instead of Observations 3.13 and 3.14 □

Proposition 3.29 finally allows us to prove the desired connection between sinks and pre-sinks for y:

Proposition 3.30 Let $S \in \mathcal{S}(y)$ and let $v \in S$ be the node with highest geodesic height among all nodes in S. Then $S = \mathrm{PS}(v)$.

Proof. The proof is completely analogous to the proof of Proposition 3.16, but uses Proposition 3.29 instead of Proposition 3.15. □

Similar to the case of Proposition 3.16, the proof of Proposition 3.30 also yields the following useful corollary:

Corollary 3.31 Let $S \in \mathcal{S}(y)$ and $u, v \in S$. Then $\mathrm{wl}(v) + \mathrm{gh}(v) = \mathrm{wl}(u) + \mathrm{gh}(u)$.

3.4.3.2 Characterization of Flooded Pre-Sinks

Using Propositions 3.16 and 3.30 together with Corollaries 3.17 and 3.31, it remains to show that $\mathcal{S}(x) = \mathcal{S}(y)$ and that, for each node $v \in V$ with $\mathrm{PS}(v) \in \mathcal{S}(x)$, it holds that $x.\mathrm{wl}(v) = y.\mathrm{wl}(v)$. To this end, we provide a characterization, for both of x and y, when all nodes in a pre-sink are flooded, in which case the corresponding pre-sink will also be called *flooded.* To this end, the reader is advised to recall the terms *threshold* of a pre-sink provided in Definition 3.6 and *positive contribution to the excess* of a pre-sink provided in Definition 3.8.

The aim in this section is to show that a pre-sink is flooded if and only if the amount of rain on the pre-sink plus the inflow from uphill nodes, which is exactly the positive contribution to the excess of the pre-sink, exceeds its threshold.

Before we prove this characterization separately for x and y, we show a simpler characterization of when a pre-sink is flooded.

Observation 3.32 Let $v \in V$. Then all nodes in $\mathrm{PS}(v)$ are flooded if and only if v is flooded.

Proof. The forward direction is clear. For the backward direction, let v be flooded. This means that $v \in S$ for some $S \in \mathcal{S}(x)$. Due to Proposition 3.15, there exists a node $u \in V$ such that $S = \mathrm{PS}(u)$. If $u = v$, this proves the claim. In the other case, it must hold that $\mathrm{gh}(u) > \mathrm{gh}(v)$. Since $v \in S = \mathrm{PS}(u)$, it holds that $\mathrm{PS}(v) \subsetneq \mathrm{PS}(u) = S$ due to Lemma 3.4, which proves the claim. The proof for $S \in \mathcal{S}(y)$ is along the same lines except for using Proposition 3.29 instead of Proposition 3.15. □

Characterization of Flooded Pre-Sinks in x

We again start by presenting the characterization for x and omit the "$x.$" when referring to variables whenever this does not lead to any confusion.

To this end, we start by proving that any water that enters a non-flooded pre-sink remains in this pre-sink. To this end, we define the *excess of a pre-sink* $\mathrm{PS}(v)$ for $v \in V$ as $\mathrm{excess}(\mathrm{PS}(v)) := \sum_{v' \in \mathrm{PS}(v)} \mathrm{excess}(v')$.

Lemma 3.33 Let $v \in V$ be non-flooded. Then, it holds that

$$\mathrm{pce}(\mathrm{PS}(v)) = \sum_{v' \in \mathrm{PS}(v)} \mathrm{excess}(v') = \mathrm{excess}(\mathrm{PS}(v)).$$

Proof. The latter equality is clear by definition. To prove the former equality, we reformulate the excess of the pre-sink according to Constraint (1):

$$\sum_{v' \in \mathrm{PS}(v)} \mathrm{excess}(v') = \sum_{v' \in \mathrm{PS}(v)} \left[\sum_{r \in \delta^-_{G^{\mathrm{ex}}}(v')} f(r) - \sum_{r \in \delta^+_{G^{\mathrm{ex}}}(v')} f(r) \right] + \sum_{v' \in \mathrm{PS}(v)} \mathrm{rain} \cdot \mathrm{area}(v')$$

We investigate the first sum and observe:

1. The flow on any arc $r \in R^{\mathrm{ex}}$ with $\alpha(r), \omega(r) \in \mathrm{PS}(v)$ appears exactly once in each of the two inner sums. Therefore, the flow on this arc does not contribute to the overall value of the outer sum.
2. The flow on any arc $r \in \delta^-_{G^{\mathrm{ex}}}(\mathrm{PS}(v))$ appears exactly once in the first inner sum.
3. We claim that any arc $r \in \delta^+_{G^{\mathrm{ex}}}(\mathrm{PS}(v))$ has $f(r) = 0$. This holds since $\mathrm{gh}(\omega(r)) > \mathrm{gh}(v)$ (otherwise, $\omega(r) \in \mathrm{PS}(v)$) and $\mathrm{gh}(\alpha(r)) + \mathrm{wl}(\alpha(r)) \leqslant \mathrm{gh}(v)$ since, otherwise, v would be flooded due to Proposition 3.15. Constraints (21) and (23) then force $\mathrm{full}(r)$ to be zero and, hence, $f(r) = 0$.
4. Any other arc does not appear in the sum at all.

Therefore, we obtain that

$$\sum_{v' \in \mathrm{PS}(v)} \mathrm{excess}(v') = \sum_{r \in \delta^-_{G^{\mathrm{ex}}}(\mathrm{PS}(v))} f(r) + \sum_{v' \in \mathrm{PS}(v)} \mathrm{rain} \cdot \mathrm{area}(v') = \mathrm{pce}(\mathrm{PS}(v)).$$

□

Using this lemma, we prove the first direction of the desired characterization of flooded pre-sinks:

Lemma 3.34 Let $v \in V$. If $\mathrm{pce}(\mathrm{PS}(v)) > \mathrm{thr}(\mathrm{PS}(v))$, then $\mathrm{PS}(v)$ is flooded.

Proof. Due to Observation 3.32, it suffices to show that v is flooded. For the sake of a contradiction, suppose that v is non-flooded. It holds that

$$\sum_{v' \in \mathrm{PS}(v)} (\mathrm{gh}(v) - \mathrm{gh}(v')) \cdot \mathrm{area}(v') = \mathrm{thr}(\mathrm{PS}(v)) < \mathrm{pce}(\mathrm{PS}(v))$$

$$\overset{\text{Lemma 3.33}}{=} \sum_{v' \in \mathrm{PS}(v)} \mathrm{excess}(v') = \sum_{v' \in \mathrm{PS}(v)} \mathrm{wl}(v') \cdot \mathrm{area}(v').$$

In order to satisfy the above inequality, a node $\hat{v} \in \mathrm{PS}(v)$ with $\mathrm{wl}(\hat{v}) > \mathrm{gh}(v) - \mathrm{gh}(\hat{v})$ must exist. Proposition 3.15 then implies that $\mathrm{wl}(v) > 0$, which yields the desired contradiction. $\square$

Before proving the other direction, we firstly observe that, for every node $v \in V$, it holds that $\mathrm{pce}(\mathrm{PS}(v)) \geqslant \mathrm{excess}(\mathrm{PS}(v))$ since, by definition, $\mathrm{pce}(\mathrm{PS}(v))$ contains all positive summands from the definition of $\mathrm{excess}(\mathrm{PS}(v))$. This observation allows proving the other direction.

Lemma 3.35 Let $v \in V$. If $\mathrm{pce}(\mathrm{PS}(v)) \leqslant \mathrm{thr}(\mathrm{PS}(v))$, then $\mathrm{PS}(v)$ is not flooded.

Proof. Again, due to Observation 3.32, it suffices to show that v is not flooded. Similar to the proof of Lemma 3.34, it holds that

$$\begin{aligned}\sum_{v' \in \mathrm{PS}(v)} (\mathrm{gh}(v) - \mathrm{gh}(v')) \cdot \mathrm{area}(v') &= \mathrm{thr}(\mathrm{PS}(v)) \\ &\geqslant \mathrm{pce}(\mathrm{PS}(v)) \\ &\geqslant \sum_{v' \in \mathrm{PS}(v)} \mathrm{excess}(v') \\ &= \sum_{v' \in \mathrm{PS}(v)} \mathrm{wl}(v') \cdot \mathrm{area}(v').\end{aligned}$$

For the sake of a contradiction, suppose that $\mathrm{wl}(v) > 0$. Applying Proposition 3.15 for every $v' \in \mathrm{PS}(v)$ yields that $\mathrm{wl}(v') > \mathrm{gh}(v) - \mathrm{gh}(v')$, which is a contradiction to the above inequality. $\square$

Lemmas 3.34 and 3.35 are summarized in the following proposition.

Proposition 3.36 Let $v \in V$. Then $\mathrm{PS}(v)$ is flooded in x if and only if it holds that $x.\mathrm{pce}(\mathrm{PS}(v)) > \mathrm{thr}(\mathrm{PS}(v))$.

Characterization of Flooded Pre-Sinks in y

We now present the analogous characterization for y and omit the "$y.$" when referring to variables whenever this does not lead to any confusion. The proof, however, is remarkably more technical than the one for x. A major part of the proof involves showing that, for each $v \in \overline{V}$, it holds that $y.\mathrm{excess}(\mathrm{PS}(v)) = \sum_{v' \in \mathrm{PS}(v)} y.\mathrm{wl}(v') \cdot \mathrm{area}(v')$, which is the first milestone of this section.

While this statement seems obvious at first glance, note that the excess of a node $v \in V$ starts building up over the whole area of nodes in $\mathrm{repres}(v)$ as soon as v becomes a leaf during the algorithm and ends building up as soon as the node leaves the graph or the algorithm terminates. Hence, the excess of v corresponds to the volume of the geometrical body, whose base area is the union of the shapes of the nodes in $\mathrm{repres}(v)$ and whose height is either the height difference of v to the next highest node in the sink, or the difference of the water level at v and the geodesic height of v if v is the highest node in the sink. This is illustrated in Figure 3.6, which can also be seen as an informal argument for the correctness of the claim.

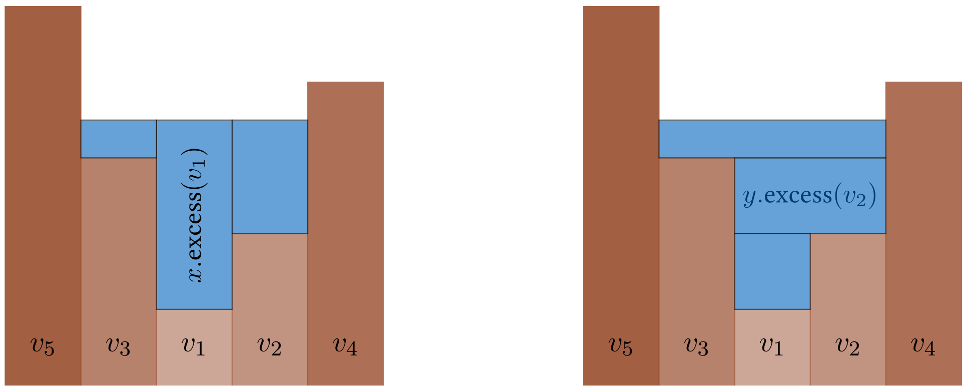


Figure 3.6.: An illustration of the excesses in the solution x on the left hand side and y on the right hand side.

We start by proving that, when the end node of an arc $r \in R$ is changed during the joinNodes-routine, its original end node is in the pre-sink of the new end node. To this end, we first introduce a notation that keeps track of changes of arcs during the algorithm.

Definition 3.37 For $t \in \{1, \dots, T\}$ and $r \in \tilde{R}_t$, we call the unique arc $r' \in R$ that r stems from the *original arc* of r and denote it by $\mathrm{oa}(r)$. Conversely, we call r the *changed arc of* r' *at iteration* t and write $\mathrm{ca}_t(r') = r$.

Note that $\mathrm{ca}_t(r)$ is not necessarily defined for each arc $r \in R$ in each iteration $t \in \{1, \dots, T\}$. We now prove the aforementioned observation.

Observation 3.38 Let $r \in R$ and $t \in \{1, \dots, T\}$ such that $\mathrm{ca}_t(r)$ is defined. Then $\omega(r) \in \mathrm{PS}(\omega(\mathrm{ca}_t(r)))$.

Proof. For the sake of a contradiction, suppose that there exist $r \in R$ and $t \in \{1, \dots, T\}$ such that $\mathrm{ca}_t(r)$ is defined and $\omega(r) \notin \mathrm{PS}(\omega(\mathrm{ca}_t(r)))$. Without loss of generality, we may assume that r is chosen such that $\alpha(r)$ has minimal geodesic height among all arcs with this property and that $\omega(r) \in \mathrm{PS}(\omega(\mathrm{ca}_{t-1}(r)))$.[17]

To enhance readability, we name the nodes as follows. The start node of the arcs $r, \mathrm{ca}_t(r)$, and $\mathrm{ca}_{t-1}(r)$ is called u.[18] The end nodes of $r, \mathrm{ca}_t(r)$, and $\mathrm{ca}_{t-1}(r)$ are called v, w, and w', respectively.

The second assumption implies that joinNodes(w', w) has been called in iteration $t-1$. Hence, there exists an arc $\hat{r} \in \tilde{R}_{t-1}$ from w to w'. Further, it holds that $\alpha(\mathrm{oa}(\hat{r})) = w$. Due to the first assumption, the claim of the observation holds true for $\mathrm{oa}(\hat{r})$, which implies that $\omega(\mathrm{oa}(\hat{r})) \in \mathrm{PS}(\omega(\mathrm{ca}_t(\mathrm{oa}(\hat{r})))) = \mathrm{PS}(\omega(\hat{r})) = \mathrm{PS}(w')$ for all $t \in \{1, \dots, T\}$. By definition of a pre-sink, it also holds that $\omega(\mathrm{oa}(\hat{r})) \in \mathrm{PS}(\alpha(\mathrm{oa}(\hat{r}))) = \mathrm{PS}(w)$. It, therefore, must hold that $\omega(\mathrm{oa}(\hat{r})) \in \mathrm{PS}(w') \cap \mathrm{PS}(w)$ and Lemma 3.4 implies that $\mathrm{PS}(w') \subseteq \mathrm{PS}(w)$. As $v \in \mathrm{PS}(w')$, it

[17]Note that $t \geqslant 2$ as $\tilde{R}_1 = R$.

[18]Note that the start node of an arc is never changed in Algorithm 2, so it holds that $\alpha(r) = \alpha(\mathrm{ca}_t(r)) = \alpha(\mathrm{ca}_{t-1}(r))$.

then also holds that $\omega(r) = v \in \mathrm{PS}(w') \subseteq \mathrm{PS}(w) = \mathrm{PS}(\omega(\mathrm{ca}_t(r)))$, which yields the desired contradiction. □

This observation can be utilized to obtain a useful result about the nodes in the sets $\mathrm{repres}(v)$ for $v \in V$ at the end of the algorithm.

Lemma 3.39 Let $v \in V$ and $t \in \{1, \dots, T\}$. Then $\mathrm{repres}_t(v) \subseteq \mathrm{PS}(v)$.

Proof. We prove this by induction over the iterations. In the first iteration, the claim clearly holds true since $\mathrm{repres}_1(v) = \{v\} \subseteq \mathrm{PS}(v)$ for all $v \in V$. Now let the claim hold in some iteration $t \in \{1, \dots, T-1\}$, i.e., $\mathrm{repres}_t(v) \subseteq \mathrm{PS}(v)$ for all $v \in V$. Let $u, w \in V$ such that $\mathtt{joinNodes}(u, w)$ is called in iteration t. For any node $v \neq w$, the set $\mathrm{repres}_t(v)$ remains unchanged in iteration t, so the claim still holds in the next iteration $t+1$. Due to the update rule in the joinNodes-routine, it remains to show that $\mathrm{repres}_t(u) \subseteq \mathrm{PS}(w)$. From the induction hypothesis, we already know that $\mathrm{repres}_t(u) \subseteq \mathrm{PS}(u)$. Further, we know that $\mathrm{gh}(u) < \mathrm{gh}(w)$. Using Lemma 3.4, it suffices to show that $\mathrm{PS}(u) \cap \mathrm{PS}(w) \neq \emptyset$. Since $\mathtt{joinNodes}(u, w)$ is called in iteration t, there must be an arc $r \in \tilde{R}_t$ with $\alpha(r) = w$ and $\omega(r) = u$. Using Observation 3.38, we get that $\omega(\mathrm{oa}(r)) \in \mathrm{PS}(u)$. By definition of a pre-sink, it also holds that $\omega(\mathrm{oa}(r)) \in \mathrm{PS}(\alpha(r)) = \mathrm{PS}(w)$. Hence, it holds that $\mathrm{PS}(u) \cap \mathrm{PS}(w) \neq \emptyset$, which completes the proof. □

If v has already left the graph in some iteration t, we can even show a stronger statement.

Lemma 3.40 Let $t \in \{1, \dots, T\}$ and $v \in V \backslash \tilde{V}_t$. Then $\mathrm{repres}_t(v) = \mathrm{PS}(v)$.

Proof. We already showed $\mathrm{repres}_t(v) \subseteq \mathrm{PS}(v)$ in Lemma 3.39. For the other direction, let $w \in \mathrm{PS}(v)$. This means there exists an undirected path with trace $(v, \tilde{w}_1, \dots, \tilde{w}_k, w)$ of nodes in $\mathrm{PS}(v)$. As $v \notin \tilde{V}_t$, it holds that $\widetilde{\mathrm{wl}}_t(v) > 0$. Applying Lemmas 3.24 and 3.27 inductively on the path yields $w \in \mathrm{repres}_t(v)$, which proves the claim. □

The proof immediately shows another result.

Corollary 3.41 If $v \in V$ is a leaf in $\tilde{G}_t$ for some $t \in \{1, \dots, T\}$, then $\mathrm{repres}_t(v) = \mathrm{PS}(v)$.

We use this statement to show that, whenever two nodes are joined, the higher one is the lowest parent of the pre-sink induced by the lower one.

Observation 3.42 Let $\mathtt{joinNodes}(u, v)$ be called during the algorithm. Then it holds that $v = \mathrm{lp}(\mathrm{PS}(u))$.

Proof. We first prove that v is a parent of $\mathrm{PS}(u)$. Let $\mathtt{joinNodes}(u, v)$ be called in iteration t. This means that there exists an arc $r \in \tilde{R}_t$ with $\alpha(r) = v$ and $\omega(r) = u$. Due to Observation 3.38, it holds that $\omega(\mathrm{oa}(r)) \in \mathrm{PS}(u)$. As the source node of an arc is never changed, it further holds that $\alpha(\mathrm{oa}(r)) = v$, which means that v is a parent of $\mathrm{PS}(u)$.

We conclude the proof by showing that v is the *lowest* parent of $\mathrm{PS}(u)$. As $\mathtt{joinNodes}(u, v)$ is called in iteration t, it holds that $u \notin \overline{V}$. Lemma 3.40 then implies that $\mathrm{repres}(u) = \mathrm{PS}(u)$.

This means that, for every arc $r \in \delta^-_G(\mathrm{PS}(u))$, if $\mathrm{ca}_t(r)$ exists, it holds that $\omega(\mathrm{ca}_t(r)) = u$. As v is by choice of the algorithm the lowest parent of u in $\tilde{G}_t$, it also is the lowest parent of $\mathrm{PS}(u)$ in G, which proves the claim. □

A further consequence of Lemma 3.40 is stated in the following corollary.

Corollary 3.43 Let $v \in V \setminus \overline{V}$. Then $\mathrm{excess}(v) = \sum_{v' \in \mathrm{PS}(v)} (\mathrm{gh}(\mathrm{lp}(\mathrm{PS}(v))) - \mathrm{gh}(v)) \cdot \mathrm{area}(v')$.

Proof. Due to Lemma 3.40, it holds that $\mathrm{repres}(v) = \mathrm{PS}(v)$. As $v \notin \overline{V}$, the node v must leave the graph in some iteration t_v. In order for the node to leave the graph, it must become the first flooded leaf, which only happens if

$$\mathrm{excess}(v)/\mathrm{area}_{t_v}(v) = \mathrm{gh}(\mathrm{lp}(\mathrm{PS}(v))) - \mathrm{gh}(v).$$

Rearranging and plugging in $\mathrm{area}_{t_v}(v) = \sum_{v' \in \mathrm{PS}(v)} \mathrm{area}(v')$ yields the desired result. □

We next investigate the structure of the sets $\mathrm{repres}(v)$ for $v \in V$ in more detail. For the following proofs, we advise the reader to recall the definition of a *follow-up* node provided in Lemma and Definition 3.5 and the result of Corollary 3.7, which are used to show an advanced structural result about the sets $\mathrm{repres}(v)$ for $v \in V$.

Corollary 3.44 Let $v \in V$ and $t \in \{1, \ldots, T\}$. Then there exists a set $U \subseteq \mathrm{FUN}(v)$ such that $\mathrm{repres}_t(v) = \{v\} \cup \bigcup_{u \in U} \mathrm{PS}(u)$.

Proof. Let $u \in \mathrm{FUN}(v)$. If $u \in \mathrm{repres}_t(v)$, then this implies that $u \notin \tilde{V}_t$. Lemma 3.40 implies that $\mathrm{repres}_t(u) = \mathrm{PS}(u)$. As $u \in \mathrm{repres}_t(v)$, all other nodes in $\mathrm{repres}_t(u)$ must also be in $\mathrm{repres}_t(v)$, which yields $\mathrm{PS}(u) \subseteq \mathrm{repres}_t(v)$.

If $u \notin \mathrm{repres}_t(v)$, then Observation 3.42 together with Corollary 3.7 implies that $u \in \tilde{V}_t$. This means that v cannot become the lowest parent of any of the nodes in $\mathrm{PS}(u)$, which implies that no node in $\mathrm{PS}(u)$ is in $\mathrm{repres}_t(v)$. □

We use Corollaries 3.43 and 3.44 to prove the next result.

Lemma 3.45 Let $v \in \overline{V}$ and $u \in \mathrm{FUN}(v)$ such that $u \in \mathrm{repres}(v)$. Then

$$\mathrm{excess}(\mathrm{PS}(u)) = \sum_{v' \in \mathrm{PS}(u)} (\mathrm{gh}(v) - \mathrm{gh}(v')) \cdot \mathrm{area}(v') = \mathrm{cap}(\mathrm{PS}(u)).$$

Proof. As $u \in \mathrm{repres}(v)$, it also holds that $\mathrm{PS}(u) \subseteq \mathrm{repres}(v)$ due to Corollary 3.44. This then implies that each node $w \in \mathrm{PS}(u)$ must leave the graph in some iteration, i.e., $w \notin \overline{V}$, which allows us to apply Corollary 3.43 and Lemma 3.40 for each node in $\mathrm{PS}(u)$:

$$\mathrm{excess}(\mathrm{PS}(u)) = \sum_{v' \in \mathrm{PS}(u)} \mathrm{excess}(v')$$

$$\overset{\text{Cor. 3.43}}{=} \sum_{v' \in \mathrm{PS}(u)} \sum_{\tilde{v} \in \mathrm{PS}(v')} (\mathrm{gh}(\mathrm{lp}(\mathrm{PS}(v'))) - \mathrm{gh}(v')) \cdot \mathrm{area}(\tilde{v})$$
$$\overset{\text{Lemma 3.40}}{=} \sum_{v' \in \mathrm{PS}(u)} \sum_{\tilde{v} \in \mathrm{repres}(v')} (\mathrm{gh}(\mathrm{lp}(\mathrm{PS}(v'))) - \mathrm{gh}(v')) \cdot \mathrm{area}(\tilde{v})$$

Let $v_1, \ldots, v_k \in \mathrm{PS}(u)$ be the nodes in order of ascending geodesic heights such that $v' \in \mathrm{repres}(v_i)$ for all $i \in \{1, \ldots, k\}$. This implies that $v_1 = v'$ and $v_k = u$. Further, Observation 3.42 implies that $v_{i+1} = \mathrm{lp}(\mathrm{PS}(v_i))$ for all $i \in \{1, \ldots, k-1\}$. The terms in the above sum involving $\mathrm{area}(v')$ can then be written as:

$$\begin{aligned} &[(\mathrm{gh}(\mathrm{lp}(\mathrm{PS}(u))) - \mathrm{gh}(u)) + (\mathrm{gh}(u) - \mathrm{gh}(v_{k-1})) + \cdots + (\mathrm{gh}(v_2) - \mathrm{gh}(v'))] \cdot \mathrm{area}(v') \\ =&(\mathrm{gh}(v) - \mathrm{gh}(v')) \cdot \mathrm{area}(v') \end{aligned}$$

Inserting this into the sum above yields the desired result. □

A similar idea is used to prove the following lemma.

Lemma 3.46 Let $v \in V \backslash \overline{V}$. Then it holds that

$$\mathrm{excess}(\mathrm{PS}(v)) = \sum_{v' \in \mathrm{PS}(v)} (\mathrm{gh}(\mathrm{lp}(\mathrm{PS}(v))) - \mathrm{gh}(v')) \cdot \mathrm{area}(v').$$

Proof. As $v \notin \overline{V}$, it must hold that $\mathrm{repres}(v) = \mathrm{PS}(v)$ due to Lemma 3.40. This means that, for every $w \in \mathrm{PS}(v)$, it must hold that $w \notin \overline{V}$. The rest of the proof is then along the same lines as the proof of Lemma 3.45 □

We now prove that, for $v \in \overline{V}$, the excess is obtained by summing up the product of the water level and the area over all nodes in $\mathrm{PS}(v)$.

Proposition 3.47 Let $v \in \overline{V}$. Then it holds that

$$\mathrm{excess}(\mathrm{PS}(v)) = \sum_{v' \in \mathrm{PS}(v)} \mathrm{wl}(v') \cdot \mathrm{area}(v').$$

Proof. For a non-flooded node $v' \in V$, it is clear that $\mathrm{excess}(v') = 0$. Due to Proposition 3.30 and Lemma 3.4, any sink $S \in \mathcal{S}(y)$ is either contained in $\mathrm{PS}(v)$ or disjoint from $\mathrm{PS}(v)$. Let $S_1, \ldots, S_k \in \mathcal{S}(y)$ be the sinks contained in $\mathrm{PS}(v)$ and let $u^{(1)}, \ldots, u^{(k)} \in V$ be the nodes inducing their corresponding pre-sinks, respectively. Then, we can write the excess as

$$\mathrm{excess}(\mathrm{PS}(v)) = \sum_{i=1}^{k} \mathrm{excess}(\mathrm{PS}(u^{(i)})) \qquad (\triangle)$$

We fix some $i \in \{1, \ldots, k\}$ and distinguish two cases: $u^{(i)} \in \overline{V}$ and $u^{(i)} \notin \overline{V}$.

Case 1: $u^{(i)} \in \overline{V}$. As $u^{(i)}$ is in a sink, it must hold that $\mathrm{wl}(u^{(i)}) > 0$, which means that $u^{(i)}$ is a leaf in $\overline{G}$. Corollary 3.41 then yields that $\mathrm{repres}(u^{(i)}) = \mathrm{PS}(u^{(i)})$. For any node $u' \in \mathrm{FUN}(u^{(i)})$, Lemma 3.45 yields

$$\mathrm{excess}(\mathrm{PS}(u')) = \sum_{v' \in \mathrm{PS}(u')} (\mathrm{gh}(u^{(i)}) - \mathrm{gh}(v')) \cdot \mathrm{area}(v').$$

Further, as $\mathrm{repres}(u^{(i)}) = \mathrm{PS}(u^{(i)})$, we get that

$$\mathrm{excess}(u^{(i)}) = \sum_{v' \in \mathrm{PS}(u^{(i)})} \mathrm{area}(v') \cdot \mathrm{wl}(u^{(i)}).$$

This yields

$$\begin{aligned}\mathrm{excess}(\mathrm{PS}(u^{(i)})) &= \sum_{u' \in \mathrm{FUN}(u^{(i)})} \sum_{v' \in \mathrm{PS}(u')} (\mathrm{gh}(u^{(i)}) - \mathrm{gh}(v')) \cdot \mathrm{area}(v') \\ &\quad + \sum_{v' \in \mathrm{PS}(u^{(i)})} \mathrm{area}(v') \cdot \mathrm{wl}(u^{(i)}) \\ &= \sum_{v' \in \mathrm{PS}(u^{(i)})} (\mathrm{wl}(u^{(i)}) + \mathrm{gh}(u^{(i)}) - \mathrm{gh}(v')) \cdot \mathrm{area}(v') \\ &\overset{\text{Prop. 3.29}}{=} \sum_{v' \in \mathrm{PS}(u^{(i)})} \mathrm{wl}(v') \cdot \mathrm{area}(v').\end{aligned}$$

Case 2: $u^{(i)} \notin \overline{V}$. In this case, Lemma 3.46 yields that

$$\mathrm{excess}(\mathrm{PS}(u^{(i)})) = \sum_{v' \in \mathrm{PS}(u^{(i)})} (\mathrm{gh}(\mathrm{lp}(\mathrm{PS}(u^{(i)}))) - \mathrm{gh}(v')) \cdot \mathrm{area}(v').$$

Let $\tilde{u} = \mathrm{lp}(\mathrm{PS}(u^{(i)}))$. As $\tilde{u} \notin \mathrm{PS}(u^{(i)}) = S_i$, it must hold that $\mathrm{wl}(\tilde{u}) = 0$ and, hence, that $\tilde{u} \in \overline{V}$. As $u^{(i)} \notin \overline{V}$, it must hold that $u^{(i)} \in \mathrm{repres}(\tilde{u})$. Observation 3.20 applied for $t = T$ then implies that, for each $v' \in \mathrm{PS}(u^{(i)})$, it holds that

$$\mathrm{gh}(\mathrm{lp}(\mathrm{PS}(u^{(i)}))) - \mathrm{gh}(v') = \mathrm{wl}(v'),$$

which then implies

$$\mathrm{excess}(\mathrm{PS}(u^{(i)})) = \sum_{v' \in \mathrm{PS}(u^{(i)})} \mathrm{wl}(v') \cdot \mathrm{area}(v'),$$

which is the same result as in the previous case.

Resubstituting this into ($\triangle$) yields

$$\mathrm{excess}(\mathrm{PS}(v)) = \sum_{v' \in \mathrm{PS}(v)} \mathrm{wl}(v') \cdot \mathrm{area}(v').$$

□

Now that we have reached this milestone, we next prove the analogue of Proposition 3.36 for y. To this end, we firstly define the positive contribution to the excess for y.

Definition 3.48 Let $v \in V$ and $t \in \{1, \dots, T\}$. We define the *positive contribution to the excess* of pre-sink $\mathrm{PS}(v)$ in iteration t as

$$y.\mathrm{pce}_t(\mathrm{PS}(v)) := \sum_{\substack{r \in \delta_G^-(\mathrm{PS}(v)):\\ \mathrm{ca}_t(r) \text{ exists}}} f_t(\mathrm{ca}_t(r)) + \sum_{v' \in \mathrm{PS}(v)} \mathrm{area}(v') \cdot \mathrm{rain} \cdot p_t$$

and the *positive contribution to the excess of pre-sink* $\mathrm{PS}(v)$ *until iteration* t as

$$y.\mathrm{pce}_{\leqslant t}(\mathrm{PS}(v)) := \sum_{t'=1}^{t} y.\mathrm{pce}_{t'}(\mathrm{PS}(v)).$$

As a short-hand notation, we define $y.\mathrm{pce}(\mathrm{PS}(v)) := y.\mathrm{pce}_{\leqslant T}(\mathrm{PS}(v))$. Further, for an iteration $t \in \{1, \dots, T\}$, the *change of excess* in t is defined as $\Delta\mathrm{excess}_t(v) := \mathrm{excess}_t(v) - \mathrm{excess}_{t-1}(v)$, where $\mathrm{excess}_0(v) = 0$ for all $v \in V$.

We start by proving a structural result, whose statement is similar to the one of Observation 3.38.

Lemma 3.49 Let $t \in \{1, \dots, T\}$ and $v \in \tilde{V}_t$. Further let $r \in R$ such that $\alpha(r) \notin \mathrm{PS}(v)$ and $\omega(r) \in \mathrm{PS}(v)$. Then $\mathrm{ca}_{t'}(r)$ exists for every $t' \leqslant t$ and $\omega(\mathrm{ca}_{t'}(r)) \in \mathrm{PS}(v)$.

Proof. We start by proving that $\mathrm{ca}_{t'}(r)$ exists for every $t' \leqslant t$. Suppose for the sake of a contradiction that this is not the case. Then there exists a last iteration $\hat{t} \leqslant t$, in which $\mathrm{ca}_{\hat{t}}(r)$ exists. This means that joinNodes$(\omega(\mathrm{ca}_{\hat{t}}(r)), \alpha(r))$ is called in iteration $\hat{t}$. As $\alpha(r) \notin \mathrm{PS}(v)$, it must hold that $\mathrm{gh}(\alpha(r)) > \mathrm{gh}(v)$. Further, as $\alpha(r)$ is a parent of $\mathrm{PS}(v)$, it holds that $v \in \mathrm{PS}(\alpha(r))$. As $\omega(\mathrm{ca}_{\hat{t}}(r)) \in \mathrm{repres}_t(\alpha(r))$, it also holds that $v \in \mathrm{repres}_t(\alpha(r))$ due to Corollary 3.44. This, however, implies that $v \notin \tilde{V}_t$, which is a contradiction. The proof of the second claim is along the same lines as the proof of Observation 3.38 and is omitted here. □

The lemma above allows proving that, as long as a node $v \in V$ has not left the graph, the change of the excess of its induced pre-sink in an interation t is exactly the positive contribution to the excess of the pre-sink in iteration t.

Lemma 3.50 Let $t \in \{1, \dots, T\}$ and $v \in \tilde{V}_t$. Then it holds that $\mathrm{pce}_t(\mathrm{PS}(v)) = \Delta\mathrm{excess}_t(\mathrm{PS}(v))$.

Proof. Firstly, note that any water that enters a node $v' \in \mathrm{PS}(v)$ or that arises from the rain on v' is sent over the outgoing arcs of v' in $\tilde{G}_t$ if there are any. Observation 3.38 guarantees that the child of such an arc is itself in $\mathrm{PS}(v')$ and, hence, in $\mathrm{PS}(v)$. Therefore, the water that arrives at the leaves of $\tilde{G}_t$ is exactly the inflow into the pre-sink plus the rain on the pre-sink.

The rain on the pre-sink in iteration t is exactly $\sum_{v' \in \mathrm{PS}(v)} \mathrm{area}(v') \cdot \mathrm{rain} \cdot \hat{p}_t$ and the inflow into the pre-sink is the flow on the arcs in $\delta_{\tilde{G}_t}^-(\mathrm{PS}(v)) = \{r \in \delta_G^-(\mathrm{PS}(v)) \mid \mathrm{ca}_t(r) \text{ exists}\}$ due to Lemma 3.49 and Observation 3.38, which proves the claim. □

There are two useful corollaries that immediately follow from this.

Corollary 3.51 Let $v \in V$ and $t \in \{1, \ldots, T\}$ such that $v \in \tilde{V}_t$. Then it holds that $\text{pce}_{\leqslant t}(\text{PS}(v)) = \text{excess}_t(\text{PS}(v))$.

Proof. Sum over the iterations from one to t and use Lemma 3.50. □

Corollary 3.52 Let $v \in V$. Then it holds that $\text{pce}(\text{PS}(v)) \geqslant \text{excess}(\text{PS}(v))$.

We can now prove the first direction of the characterization when a pre-sink is flooded.

Lemma 3.53 Let $v \in V$. If $\text{pce}(\text{PS}(v)) > \text{thr}(\text{PS}(v))$, then $\text{PS}(v)$ is flooded.

Proof. Due to Observation 3.32, it suffices to show that v is flooded. For the sake of a contradiction, suppose that v is not flooded. In particular, this means that $v \in \overline{V}$. Corollary 3.51 then implies that

$$\begin{aligned}\sum_{v' \in \text{PS}(v)} (\text{gh}(v) - \text{gh}(v')) \cdot \text{area}(v') = \text{thr}(\text{PS}(v)) < \text{pce}(\text{PS}(v)) = \text{excess}(\text{PS}(v))\\ \overset{\text{Prop. 3.47}}{=} \sum_{v' \in \text{PS}(v)} \text{wl}(v') \cdot \text{area}(v').\end{aligned}$$

This means that there exists a node $v' \in \text{PS}(v)$ with $\text{wl}(v') > (\text{gh}(v) - \text{gh}(v'))$. Then Proposition 3.29 implies that $\text{wl}(v) > 0$, which yields the desired contradiction. □

Next, the other direction is shown.

Lemma 3.54 Let $v \in V$. If $\text{pce}(\text{PS}(v)) \leqslant \text{thr}(\text{PS}(v))$, then $\text{PS}(v)$ is not flooded.

Proof. Due to Observation 3.32, it suffices to show that v is not flooded. For the sake of a contradiction, suppose that v is flooded. We distinguish two cases:

<u>Case 1:</u> $v \notin \overline{V}$

Due to Lemma 3.46, it holds that

$$\text{excess}(\text{PS}(v)) = \sum_{v' \in \text{PS}(v)} (\text{gh}(\text{lp}(\text{PS}(v))) - \text{gh}(v')) \cdot \text{area}(v').$$

Corollary 3.52 then implies that

$$\begin{aligned}\sum_{v' \in \text{PS}(v)} (\text{gh}(v) - \text{gh}(v')) \cdot \text{area}(v') = \text{thr}(\text{PS}(v)) \geqslant \text{pce}(\text{PS}(v)) \geqslant \text{excess}(\text{PS}(v))\\ = \sum_{v' \in \text{PS}(v)} (\text{gh}(\text{lp}(\text{PS}(v))) - \text{gh}(v')) \cdot \text{area}(v').\end{aligned}$$

This, however, is a contradiction to $\text{gh}(v) < \text{gh}(\text{lp}(\text{PS}(v)))$.

Case 2: $v \in \overline{V}$
Proposition 3.47 yields

$$\operatorname{excess}(\operatorname{PS}(v)) = \sum_{v' \in \operatorname{PS}(v)} \operatorname{wl}(v') \cdot \operatorname{area}(v').$$

Using Corollary 3.52, we get that

$$\begin{aligned}\sum_{v' \in \operatorname{PS}(v)} (\operatorname{gh}(v) - \operatorname{gh}(v')) \cdot \operatorname{area}(v') = \operatorname{thr}(\operatorname{PS}(v)) \geqslant \operatorname{pce}(\operatorname{PS}(v)) &\geqslant \operatorname{excess}(\operatorname{PS}(v)) \\ &= \sum_{v' \in \operatorname{PS}(v)} \operatorname{wl}(v') \cdot \operatorname{area}(v').\end{aligned}$$

Proposition 3.29, however, states that $\operatorname{gh}(v') + \operatorname{wl}(v') = \operatorname{gh}(v) + \operatorname{wl}(v) > \operatorname{gh}(v)$ for each $v' \in \operatorname{PS}(v)$, which is a contradiction. □

As a result, we obtain the desired characterization of flooded pre-sinks for y:

Proposition 3.55 Let $v \in V$. Then $\operatorname{PS}(v)$ is flooded in y if and only if it holds $y.\operatorname{pce}(\operatorname{PS}(v)) > \operatorname{thr}(\operatorname{PS}(v))$.

3.4.3.3 Characterization of Inflows and Outflows of Pre-Sinks

Propositions 3.36 and 3.55 imply that, if $x.\operatorname{pce}(\operatorname{PS}(v)) = y.\operatorname{pce}(\operatorname{PS}(v))$ for a node $v \in V$, then $\operatorname{PS}(v)$ is flooded in x if and only if it is flooded in y. The fact that the positive contribution to the excess depends on the flows in the graph motivates to investigate the flows in more detail. In this section, we present a characterization of the inflows and outflows of pre-sinks.

Characterization of Infows and Outflows of Pre-Sinks for x

We start by presenting the characterization for x and omit the "$x.$" when referring to variables whenever this does not lead to any confusion.

Definition 3.56 Let $S \in \mathcal{S}(x)$. The sink S is called *filled to capacity (ftc)* if $x.\operatorname{excess}(S) = \operatorname{cap}(S)$, and it is called *backfloating* if

$$\sum_{\substack{r \in R^{\text{ex}}: \\ \alpha(r) \in S, \\ \omega(r) = \operatorname{lp}(S)}} x.f(r) > \sum_{\substack{r \in R^{\text{ex}}: \\ \alpha(r) = \operatorname{lp}(S), \\ \omega(r) \in S}} x.f(r).$$

In this case, the sink's *backfloat* is defined as

$$x.\operatorname{bf}(S) := \sum_{\substack{r \in R^{\text{ex}}: \\ \alpha(r) \in S, \\ \omega(r) = \operatorname{lp}(S)}} x.f(r) - \sum_{\substack{r \in R^{\text{ex}}: \\ \alpha(r) = \operatorname{lp}(S), \\ \omega(r) \in S}} x.f(r).$$

The following lemma provides an alternative characterization of a pre-sink being ftc.

Lemma 3.57 Let $S \in \mathcal{S}(x)$. Then S is ftc if and only if $\mathrm{wl}(v) + \mathrm{gh}(v) = \mathrm{gh}(\mathrm{lp}(S))$ for all $v \in S$.

Proof. Let $S \in \mathcal{S}(x)$ be ftc. Using Constraint (2) together with the definition of $\mathrm{cap}(S)$, this is equivalent to

$$\sum_{v \in S} \mathrm{wl}(v) \cdot \mathrm{area}(v) = \sum_{v \in S} (\mathrm{gh}(\mathrm{lp}(S) - \mathrm{gh}(v)) \cdot \mathrm{area}(v).$$

Proposition 3.15 further implies that $\mathrm{wl}(v) \leqslant \mathrm{gh}(\mathrm{lp}(S) - \mathrm{gh}(v)$ for all $v \in S$ since otherwise $\mathrm{wl}(\mathrm{lp}(S)) > 0$, which is a contradiction to the structure of a sink. This means that the above equation holds if and only if $\mathrm{wl}(v) + \mathrm{gh}(v) = \mathrm{gh}(\mathrm{lp}(S))$ for all $v \in S$. □

Corollary 3.58 Let $S \in \mathcal{S}(x)$ be non-ftc. Then $\mathrm{wl}(v) + \mathrm{gh}(v) < \mathrm{gh}(\mathrm{lp}(S))$ for all $v \in S$.

Proof. Use Lemma 3.57 and Corollary 3.17. □

These two statements allow investigating the flows on arcs leading into a sink in more detail.

Corollary 3.59 Let $S \in \mathcal{S}(x)$ be ftc. Then every arc $r \in \delta_G^-(S) \cap \delta_G^+(\mathrm{lp}(S))$ is full and every arc $r \in \delta_G^-(S) \backslash \delta_G^+(\mathrm{lp}(S))$ is not full.

Proof. Due to Lemma 3.57, it holds that $\mathrm{wl}(v) + \mathrm{gh}(v) = \mathrm{gh}(\mathrm{lp}(S))$ for all $v \in S$. Let now $r \in \delta_G^-(S) \cap \delta_G^+(\mathrm{lp}(S))$, then r is full due to Constraints (22) and (24). For an arc $r \in \delta_G^-(S) \backslash \delta_G^+(\mathrm{lp}(S))$, it must hold that $\mathrm{gh}(\alpha(r))) > \mathrm{gh}(\mathrm{lp}(S)) = \mathrm{wl}(v) + \mathrm{gh}(v)$. Hence, Constraint (36) implies that r is not full. □

Corollary 3.60 Let $S \in \mathcal{S}(x)$ be non-ftc. Then no arc $r \in \delta_G^-(S)$ is full.

Proof. Use Corollary 3.58 and Constraint (36). □

Next, we provide a characterization of a sink being backfloating. The idea is that, if the rain on the sink plus the flow on incoming arcs from nodes that are not the lowest parent of the sink is larger than the sink's capacity, then the sink is backfloating. At this point, we advise the reader to recall the definitions of $x.\mathrm{pcelp}(\mathrm{PS}(v))$ and $x.\mathrm{pcenlp}(\mathrm{PS}(v))$ provided in Definition 3.9. We firstly observe the following:

Observation 3.61 Let $S \in \mathcal{S}(x)$ be backfloating, then it is also ftc.

Proof. Suppose S is not ftc. Then, due to Corollary 3.60, all incoming arcs into S are not full, which means that

$$\sum_{\substack{r \in R^{\mathrm{ex}}: \\ \alpha(r) \in S, \\ \omega(r) = \mathrm{lp}(S)}} x.f(r) = 0.$$

Hence, S cannot be backfloating. □

We start proving the first direction of our characterization.

Lemma 3.62 Let $S \in \mathcal{S}(x)$ be backfloating. Then it holds that $x.\text{pcenlp}(S) > \text{cap}(S)$ and $x.\text{bf}(S) = x.\text{pcenlp}(S) - \text{cap}(S)$.

Proof. Due to Observation 3.61, it holds that S is ftc. Further, Corollary 3.59 implies that every arc $r \in \delta^-_G(S) \cap \delta^+_G(\text{lp}(S))$ is full and every arc $r \in \delta^-_G(S) \backslash \delta^+_G(\text{lp}(S))$ is not full, which implies that:

$$\begin{aligned}
\text{cap}(S) &= \text{excess}(S)\\
&= \sum_{v' \in S}\left[\sum_{r \in \delta^-_{G^\text{ex}}(v')} x.f(r) - \sum_{r \in \delta^+_{G^\text{ex}}(v')} x.f(r)\right] + \sum_{v' \in S} \text{rain} \cdot \text{area}(v')\\
&= \sum_{r \in \delta^-_{G^\text{ex}}(S)} x.f(r) + \sum_{v' \in S} \text{rain} \cdot \text{area}(v') - \sum_{r \in \delta^+_{G^\text{ex}}(S)} x.f(r)\\
&\overset{\text{Cor. 3.59}}{=} \sum_{\substack{r \in \delta^-_{G^\text{ex}}(S):\\ \alpha(r) \neq \text{lp}(S)}} x.f(r) + \sum_{v' \in \text{PS}(v)} \text{rain} \cdot \text{area}(v') + \sum_{\substack{r \in \delta^-_{G^\text{ex}}(S):\\ \alpha(r) = \text{lp}(S)}} x.f(r) - \sum_{\substack{r \in \delta^+_{G^\text{ex}}(S):\\ \omega(r) = \text{lp}(S)}} x.f(r)\\
&= x.\text{pcenlp}(S) + \sum_{\substack{r \in \delta^-_{G^\text{ex}}(S):\\ \alpha(r) = \text{lp}(S)}} x.f(r) - \sum_{\substack{r \in \delta^+_{G^\text{ex}}(S):\\ \omega(r) = \text{lp}(S)}} x.f(r)
\end{aligned}$$

Using that

$$\sum_{\substack{r \in \delta^-_{G^\text{ex}}(S):\\ \alpha(r) = \text{lp}(S)}} x.f(r) - \sum_{\substack{r \in \delta^+_{G^\text{ex}}(S):\\ \omega(r) = \text{lp}(S)}} x.f(r) < 0,$$

it follows that $\text{cap}(S) < \text{pcenlp}(S)$ and that $\text{cap}(S) = \text{pcenlp}(S) + \text{bf}(S)$, which concludes the proof. ☐

The following corollary is a direct consequence of this proof:

Corollary 3.63 Let $S \in \mathcal{S}(x)$ be ftc. Then it holds that

$$\sum_{\substack{r \in \delta^-_{G^\text{ex}}(S):\\ \alpha(r) = \text{lp}(S)}} x.f(r) - \sum_{\substack{r \in \delta^+_{G^\text{ex}}(S):\\ \omega(r) = \text{lp}(S)}} x.f(r) = \text{cap}(S) - x.\text{pcenlp}(S).$$

Next, the opposite direction is shown.

Lemma 3.64 Let $S \in \mathcal{S}(x)$ with $x.\text{pcenlp}(S) > \text{cap}(S)$. Then S is backfloating with $x.\text{bf}(S) = x.\text{pcenlp}(S) - \text{cap}(S)$.

Proof. We first prove that S is ftc. Suppose this was not the case, then, by Corollary 3.60, no arc in $\delta^-_G(S)$ is full. Hence, it holds that

$$
\begin{aligned}
\text{excess}(S) &= \sum_{v' \in S} \left[\sum_{r \in \delta^-_{G^{\text{ex}}}(v')} x.f(r) - \sum_{r \in \delta^+_{G^{\text{ex}}}(v')} x.f(r) \right] + \sum_{v' \in S} \text{rain} \cdot \text{area}(v') \\
&= \sum_{r \in \delta^-_{G^{\text{ex}}}(S)} x.f(r) + \sum_{v' \in S} \text{rain} \cdot \text{area}(v') - \underbrace{\sum_{r \in \delta^+_{G^{\text{ex}}}(S)} x.f(r)}_{=0} \\
&= x.\text{pcenlp}(S) + x.\text{pcelp}(S) > \text{cap}(S),
\end{aligned}
$$

which is a contradiction. By the same arguments as in the proof of Lemma 3.62, the desired result is obtained. □

The two lemmas are summarized in the following proposition:

Proposition 3.65 Let $S \in \mathcal{S}(x)$. Then S is backfloating if and only if $x.\text{pcenlp}(S) > \text{cap}(S)$. In this case, it holds that $x.\text{bf}(S) = x.\text{pcenlp}(S) - \text{cap}(S)$.

Characterization of Infows and Outflows of Pre-Sinks for y

We proceed by presenting an analogue characterization for y and omit the "$y.$" when referring to variables whenever this does not lead to any confusion.

Definition 3.66 Let $S \in \mathcal{S}(y)$. We call S *filled to capacity (ftc)* if $y.\text{excess}(S) = \text{cap}(S)$. Further, let $v \in V$ such that $S = \text{PS}(v)$ and $v \notin \overline{V}$. We then call S *backfloating* if

$$
\begin{aligned}
&\sum_{t > t_v} \sum_{\substack{r \in \delta^-_G(\text{PS}(v)): \\ \text{ca}_t(r) \text{ exists}, \\ \alpha(r) \neq \text{lp}(\text{PS}(v))}} y.f_t(\text{ca}_t(r)) + \sum_{v' \in \text{PS}(v)} \text{area}(v') \cdot \text{rain} \cdot (1 - \text{sp}_{t_v}) \\
> &\sum_{t=1}^{T} \sum_{\substack{r \in \delta^-_G(\text{PS}(v)): \\ \text{ca}_t(r) \text{ exists}, \\ \alpha(r) = \text{lp}(\text{PS}(v))}} y.f_t(\text{ca}_t(r)).
\end{aligned}
$$

In this case, the sink's *backfloat* is defined by

$$
\begin{aligned}
\text{bf}(S) := &\sum_{t > t_v} \sum_{\substack{r \in \delta^-_G(\text{PS}(v)): \\ \text{ca}_t(r) \text{ exists}, \\ \alpha(r) \neq \text{lp}(\text{PS}(v))}} y.f_t(\text{ca}_t(r)) + \sum_{v' \in \text{PS}(v)} \text{area}(v') \cdot \text{rain} \cdot (1 - \text{sp}_{t_v}) \\
&- \sum_{t=1}^{T} \sum_{\substack{r \in \delta^-_G(\text{PS}(v)): \\ \text{ca}_t(r) \text{ exists}, \\ \alpha(r) = \text{lp}(\text{PS}(v))}} y.f_t(\text{ca}_t(r)).
\end{aligned}
$$

In contrast to the MIP, only one direction of the characterization of a sink being ftc is needed for proving the analogous statement of Proposition 3.65. It is worth noting that the other direction holds as well.
We start by proving the analogous statement to Corollary 3.58 for y.

Lemma 3.67 Let $S \in \mathcal{S}(y)$ be non-ftc. Then, for all $v \in S$, it holds that $\mathrm{wl}(v) + \mathrm{gh}(v) < \mathrm{gh}(\mathrm{lp}(S))$.

Proof. Clearly, it cannot hold for any $v' \in S$ that $\mathrm{wl}(v') + \mathrm{gh}(v') > \mathrm{gh}(\mathrm{lp}(S))$ as, in this case, Proposition 3.29 implies that $\mathrm{wl}(\mathrm{lp}(S)) > 0$ and, hence, that $\mathrm{lp}(S) \in S$, which is a contradiction to the structure of a sink. So suppose that $\mathrm{wl}(\tilde{v}) + \mathrm{gh}(\tilde{v}) = \mathrm{gh}(\mathrm{lp}(S))$ holds for some $\tilde{v} \in S$. Again, Proposition 3.29 implies that $\mathrm{wl}(v') + \mathrm{gh}(v') = \mathrm{gh}(\mathrm{lp}(S))$ holds for all $v' \in S$. Let $v \in V$ such that $S = \mathrm{PS}(v)$. We then distinguish two cases.

Case 1: $v \in \overline{V}$

We apply Proposition 3.47 and obtain

$$\begin{aligned}\mathrm{excess}(S) &= \sum_{v' \in S} \mathrm{wl}(v') \cdot \mathrm{area}(v') \\ &= \sum_{v' \in S} (\mathrm{gh}(\mathrm{lp}(S)) - \mathrm{gh}(v')) \cdot \mathrm{area}(v') \\ &= \mathrm{cap}(S),\end{aligned}$$

which is a contradiction to S being non-ftc.

Case 2: $v \notin \overline{V}$

We apply Lemma 3.46 and obtain

$$\mathrm{excess}(S) = \sum_{v' \in S} (\mathrm{gh}(\mathrm{lp}(S)) - \mathrm{gh}(v')) \cdot \mathrm{area}(v') = \mathrm{cap}(S),$$

which, again, is a contradiction to S being non-ftc. □

This allows proving the following observation.

Observation 3.68 Let $S \in \mathcal{S}(y)$ be backfloating, then it is also ftc.

Proof. Suppose this is not the case, then Lemma 3.67 implies that, for all $v' \in S$, it holds that $\mathrm{wl}(v') + \mathrm{gh}(v') < \mathrm{gh}(\mathrm{lp}(S))$. In particular, for $v \in V$ such that $S = \mathrm{PS}(v)$, this means that $v \in \overline{V}$, which means that S is not backfloating. □

We divide the positive contribution to the excess in a similar manner as for x.

Definition 3.69 Let $v \in V$ and $t \in \{1, \dots, T\}$. We define the *positive contribution to the excess from non-lowest parents* of pre-sink $\mathrm{PS}(v)$ until iteration t as

$$y.\mathrm{pcenlp}_t(\mathrm{PS}(v)) := \sum_{t'=1}^{t} \sum_{\substack{r \in \delta_G^-(\mathrm{PS}(v)):\\ \mathrm{ca}_{t'}(r) \text{ exists},\\ \alpha(r) \neq \mathrm{lp}(\mathrm{PS}(v))}} y.f_{t'}(\mathrm{ca}_t(r)) + \sum_{v' \in \mathrm{PS}(v)} \mathrm{area}(v') \cdot \mathrm{rain} \cdot \mathrm{sp}_t$$

and the *positive contribution to the excess from the lowest parent* of pre-sink $\mathrm{PS}(v)$ until iteration t as

$$y.\mathrm{pcelp}_t(\mathrm{PS}(v)) := \sum_{t'=1}^{t} \sum_{\substack{r \in \delta_G^-(\mathrm{PS}(v)):\\ \mathrm{ca}_{t'}(r) \text{ exists},\\ \alpha(r) = \mathrm{lp}(\mathrm{PS}(v))}} y.f_{t'}(\mathrm{ca}_t(r)).$$

Further, we introduce the notations $\mathrm{pcenlp}(\mathrm{PS}(v)) := \mathrm{pcenlp}_T(\mathrm{PS}(v))$ and $\mathrm{pcelp}(\mathrm{PS}(v)) := \mathrm{pcelp}_T(\mathrm{PS}(v))$.

Using Observation 3.68, we prove the first direction of our characterization of a sink being backfloating.

Lemma 3.70 Let $S \in \mathcal{S}(y)$ be backfloating. Then it holds that $\mathrm{pcenlp}(S) > \mathrm{cap}(S)$ and that $\mathrm{bf}(S) = \mathrm{pcenlp}(S) - \mathrm{cap}(S)$.

Proof. Let $v \in V$ such that $S = \mathrm{PS}(v)$. As S is backfloating, it must hold that $v \notin \overline{V}$ and, due to Observation 3.68, it must also hold that $\mathrm{cap}(S) = \mathrm{excess}(S)$. We then argue that

$$\begin{aligned}
\mathrm{cap}(S) &= \mathrm{excess}(S)\\
&\overset{\text{Cor. 3.51}}{=} \mathrm{pce}_{\leqslant t_v}(S)\\
&= \mathrm{pcenlp}_{t_v}(S) + \mathrm{pcelp}_{t_v}(S)\\
&= \sum_{t' \leqslant t_v} \sum_{\substack{r \in \delta_G^-(S):\\ \mathrm{ca}_{t'}(r) \text{ exists},\\ \alpha(r) \neq \mathrm{lp}(S)}} f_{t'}(\mathrm{ca}_t(r)) + \sum_{v' \in S} \mathrm{area}(v') \cdot \mathrm{rain} \cdot \mathrm{sp}_{t_v}\\
&\quad + \sum_{t' \leqslant t_v} \sum_{\substack{r \in \delta_G^-(S):\\ \mathrm{ca}_{t'}(r) \text{ exists},\\ \alpha(r) = \mathrm{lp}(S)}} f_{t'}(\mathrm{ca}_t(r)) + \underbrace{\sum_{t' > t_v} \sum_{\substack{r \in \delta_G^-(S):\\ \mathrm{ca}_{t'}(r) \text{ exists},\\ \alpha(r) = \mathrm{lp}(S)}} f_{t'}(\mathrm{ca}_t(r))}_{=0}\\
&= \sum_{t' \leqslant t_v} \sum_{\substack{r \in \delta_G^-(S):\\ \mathrm{ca}_{t'}(r) \text{ exists},\\ \alpha(r) \neq \mathrm{lp}(S)}} f_{t'}(\mathrm{ca}_t(r)) + \sum_{v' \in S} \mathrm{area}(v') \cdot \mathrm{rain} \cdot \mathrm{sp}_{t_v} + \mathrm{pcelp}(S)
\end{aligned}$$

$$= \text{pcenlp}(S) - \sum_{t>t_v} \sum_{\substack{r\in\delta^-_G(S):\\ \text{ca}_t(r) \text{ exists},\\ \alpha(r)\neq \text{lp}(S)}} f_t(\text{ca}_t(r)) - \sum_{v'\in S} \text{area}(v') \cdot \text{rain} \cdot (1-\text{sp}_{t_v}) + \text{pcelp}(S).$$

Using that S is backfloating, by definition, it follows that

$$\sum_{t>t_v} \sum_{\substack{r\in\delta^-_G(S):\\ \text{ca}_t(r) \text{ exists},\\ \alpha(r)\neq \text{lp}(S)}} f_t(\text{ca}_t(r)) + \sum_{v'\in S} \text{area}(v') \cdot \text{rain} \cdot (1-\text{sp}_{t_v}) > \text{pcelp}(S),$$

which means that $\text{cap}(S) < \text{pcelp}(S)$. Further, plugging in the formula for $\text{bf}(S)$ into the above equation yields $\text{bf}(S) = \text{pcenlp}(S) - \text{cap}(S)$. □

Next, the other direction is shown.

Lemma 3.71 Let $S \in \mathcal{S}(y)$ with $\text{pcenlp}(S) > \text{cap}(S)$. Then it holds that S is backfloating and that $\text{bf}(S) = \text{pcenlp}(S) - \text{cap}(S)$.

Proof. Let $v \in V$ such that $S = \text{PS}(v)$. We first show that S is ftc. Suppose the contrary, then it holds that $v \in \overline{V}$. Hence, applying Corollary 3.51 yields

$$\text{cap}(S) > \text{excess}(S) = \text{pce}(S) = \text{pcenlp}(S) + \text{pcelp}(S) \geqslant \text{pcenlp}(S),$$

which is a contradiction. Executing the same proof as for Lemma 3.70 yields the desired result. □

The two previous lemmas are summarized in the following proposition.

Proposition 3.72 Let $S \in \mathcal{S}(y)$. Then S is backfloating if and only if $y.\text{pcenlp}(S) > \text{cap}(S)$. In this case, it holds that $y.\text{bf}(S) = y.\text{pcenlp}(S) - \text{cap}(S)$.

A further direct consequence of the proof is the following statement.

Corollary 3.73 Let $S \in \mathcal{S}(y)$ be ftc. Then it holds that

$$\begin{aligned}\text{cap}(S) = y.\text{pcenlp}(S) - \sum_{t>t_v} \sum_{\substack{r\in\delta^-_G(S):\\ \text{ca}_t(r) \text{ exists},\\ \alpha(r)\neq \text{lp}(\text{PS}(v))}} y.f_t(\text{ca}_t(r)) \\ - \sum_{v'\in S} \text{area}(v') \cdot \text{rain} \cdot (1-\text{sp}_{t_v}) + y.\text{pcelp}(S).\end{aligned}$$

3.4.3.4 Proof of Equal Water Levels in x and y

We now use the results obtained about the water levels and flows in pre-sinks to show that $x.\text{wl}(v) = y.\text{wl}(v)$ for all $v \in V$. The idea of the proof is to show, from the highest node to the lowest node, that the water levels are the same in x and y and that this also holds for the flows on all outgoing arcs that are not full in x. The overall flows over an arc $r \in R$ in y are defined by $y.f(r) := \sum_{\substack{t \in \{1,\dots,T\}: \\ \mathrm{ca}_t(r) \text{ exists}}} y.f_t(\mathrm{ca}(r))$. We formalize the above-mentioned property of a node in the following definition:

Definition 3.74 A node $v \in V$ is called *explored above* if v is non-flooded in x and y and all $v' \in V$ with $\mathrm{gh}(v') > \mathrm{gh}(v)$ fulfill the following properties:

1. $x.\mathrm{wl}(v') = y.\mathrm{wl}(v')$
2. For each arc $r \in \delta^+_G(v')$ that is not full in x, it holds that $x.f(r) = y.f(r)$.

It is now shown that, if a node $v \in V$ is explored above, each sink S that is induced by a follow-up node of v is backfloating in x if and only if it is in y.

Lemma 3.75 Let $v \in V$ be explored above. Then for each $u \in \mathrm{FUN}(v)$, it holds that $\mathrm{PS}(u)$ is a backfloating sink in x if and only if it is in y. If it is backfloating, it further holds that $x.\mathrm{bf}(\mathrm{PS}(u)) = y.\mathrm{bf}(\mathrm{PS}(u))$.

Proof. Due to Propositions 3.36 and 3.55 and Propositions 3.65 and 3.72, it suffices to show that $x.\mathrm{pcenlp}(\mathrm{PS}(u)) = y.\mathrm{pcenlp}(\mathrm{PS}(u))$. As v is not flooded, it holds that every arc $r \in \delta^-_G(\mathrm{PS}(u))$ with $\alpha(r) \neq v$ is not full in x. Then, due to Property 2 of v being explored above, it holds that $x.\mathrm{pcenlp}(\mathrm{PS}(u)) = y.\mathrm{pcenlp}(\mathrm{PS}(u))$, which proves the claim. □

The lemma in particular shows that, if a pre-sink induced by a follow-up node u is a backfloating sink, the water levels of all nodes in $\mathrm{PS}(u)$ are the same in x and y. We next investigate the follow-up nodes whose induced pre-sinks are not ftc sinks (i.e., either not a sink or non-ftc). To this end, the reader is advised to recall the definition of the *total ratio of the lowest parent* of a pre-sink provided in Definition 3.10. We observe an important property of the total ratio of the lowest parent.

Observation 3.76 Let $v \in V$ and $z \in \{x, y\}$. Further, let $u^{(1)}, u^{(2)} \in \mathrm{FUN}(v)$ such that $\mathrm{PS}(u^{(1)})$ and $\mathrm{PS}(u^{(2)})$ are not ftc sinks in z. Then it holds:

$$\frac{z.\mathrm{pcelp}(\mathrm{PS}(u^{(1)}))}{z.\mathrm{pcelp}(\mathrm{PS}(u^{(2)}))} = \frac{\mathrm{ratiolp}(\mathrm{PS}(u^{(1)}))}{\mathrm{ratiolp}(\mathrm{PS}(u^{(2)}))}$$

Proof. For $z = x$, this is clear since all arcs in $\delta^-_G(\mathrm{PS}(u^{(1)}))$ and $\delta^-_G(\mathrm{PS}(u^{(2)}))$ are not full. The claim is then clear from Constraint (31). For $z = y$, the claim is also clear as all arcs in $\delta^-_G(\mathrm{PS}(u^{(1)}))$ and $\delta^-_G(\mathrm{PS}(u^{(2)}))$ are never removed from the graph (otherwise, $\mathrm{PS}(u^{(1)})$ or $\mathrm{PS}(u^{(2)})$ would be ftc). □

The two previous observations show that the following is well-defined:

Definition 3.77 Let $z \in \{x, y\}$ and $v \in V$ be non-flooded with $z.\mathrm{pce}(\mathrm{PS}(v)) < \mathrm{thr}(v)$. Further let $u \in \mathrm{FUN}(v)$ such that $\mathrm{PS}(u)$ is not an ftc sink in z. Then we define the *non-full arc distribution* of v as

$$z.\text{nfad}(v) := \frac{z.\text{pcelp}(\text{PS}(u))}{\text{ratiolp}(\text{PS}(u))}.$$

We next show that $x.\text{nfad}(v) = y.\text{nfad}(v)$ for each non-flooded and explored-above node $v \in V$ with $z.\text{pce}(\text{PS}(v)) < \text{thr}(v)$. To prove this, an additional lemma is required.

Lemma 3.78 Let $z \in \{x, y\}$ and $v \in V$ be non-flooded with $z.\text{pce}(\text{PS}(v)) < \text{thr}(v)$. Further let $u \in \text{FUN}(v)$. If it further holds that $z.\text{nfad}(v) \cdot \text{ratiolp}(\text{PS}(u)) \geqslant \text{cap}(\text{PS}(u)) - z.\text{pcenlp}(\text{PS}(u))$, then $\text{PS}(u)$ is an ftc sink in z.

Proof. By definition, it holds that

$$z.\text{pcelp}(\text{PS}(u)) = z.\text{nfad}(v) \cdot \text{ratiolp}(\text{PS}(u)) \geqslant \text{cap}(\text{PS}(u)) - z.\text{pcenlp}(\text{PS}(u)),$$

which means that $z.\text{excess}(\text{PS}(u)) \geqslant \text{cap}(\text{PS}(u))$ (due to Lemma 3.33 in the case of $z = x$ and Corollary 3.51 in the case of $z = y$). As v is non-flooded, it must also hold that $z.\text{excess}(\text{PS}(u)) \leqslant \text{cap}(\text{PS}(u))$, which means that equality holds and $\text{PS}(u)$ is an ftc sink. □

We proceed by showing that $x.\text{nfad}(v) = y.\text{nfad}(v)$ for each explored-above node $v \in V$ for which it holds that $z.\text{pce}(\text{PS}(v)) < \text{thr}(\text{PS}(v))$.

Lemma 3.79 Let $z \in \{x, y\}$ and $v \in V$ be explored above with $z.\text{pce}(\text{PS}(v)) < \text{thr}(\text{PS}(v))$. Then it holds that $x.\text{nfad}(v) = y.\text{nfad}(v)$.

Proof. As v is explored above, the node is non-flooded in both solutions (and in particular, it holds that $v \in \overline{V}$). Hence, it holds that

$$x.\text{excess}(\text{PS}(v)) = x.\text{pce}(\text{PS}(v)) = y.\text{pce}(\text{PS}(v)) = y.\text{excess}(\text{PS}(v)) \tag{$\triangledown$}$$

due to Lemma 3.33 and Corollary 3.51.

For the sake of a contradiction, suppose that $x.\text{nfad}(v) \neq y.\text{nfad}(v)$. It is firstly assumed that $x.\text{nfad}(v) > y.\text{nfad}(v)$. The proof of the other case is along the same lines.

For all $u \in \text{FUN}(v)$ for which $\text{PS}(u)$ is an ftc sink in y, Lemma 3.78 shows that $\text{PS}(u)$ is also an ftc sink in x. In particular, this means that $x.\text{excess}(\text{PS}(u)) = y.\text{excess}(\text{PS}(u))$. For all $u \in \text{FUN}(v)$, for which $\text{PS}(u)$ is not an ftc sink in y, we distinguish two cases.

<u>Case 1:</u> $\text{PS}(u)$ is an ftc sink in x.

Then $y.\text{excess}(\text{PS}(u)) < \text{cap}(\text{PS}(u)) = x.\text{excess}(\text{PS}(u))$ by definition of ftc.

<u>Case 2:</u> $\text{PS}(u)$ is not an ftc sink in x.

Then, by definition, it holds that

$$x.\text{pcelp}(\text{PS}(u)) = x.\text{nfad}(v) \cdot \text{ratiolp}(\text{PS}(u)) > y.\text{nfad}(v) \cdot \text{ratiolp}(\text{PS}(u)) = y.\text{pcelp}(\text{PS}(u)).$$

As v is explored above, it further holds that $x.\text{pcenlp}(\text{PS}(u)) = y.\text{pcenlp}(\text{PS}(u))$, which yields $x.\text{pce}(\text{PS}(u)) > y.\text{pce}(\text{PS}(u))$. As $\text{PS}(u)$ is not an ftc sink in both x and y, it holds that

$$x.\text{excess}(\text{PS}(u)) = x.\text{pce}(\text{PS}(u)) > y.\text{pce}(\text{PS}(u)) = y.\text{excess}(\text{PS}(u)).$$

All in all, it holds that

$$\begin{aligned} x.\text{excess}(\text{PS}(v)) &= \sum_{u \in \text{FUN}(v)} x.\text{excess}(\text{PS}(u)) \\ &> \sum_{u \in \text{FUN}(v)} y.\text{excess}(\text{PS}(u)) = y.\text{excess}(\text{PS}(v)), \end{aligned}$$

which is a contradiction to ($\triangledown$). □

We now prove the final proposition, which shows that, if a node v is explored above with $z.\text{pce}(\text{PS}(v)) < \text{thr}(v)$, then its non-flooded follow-up nodes are as well.

Proposition 3.80 Let $v \in V$ be explored above with $z.\text{pce}(\text{PS}(v)) < \text{thr}(v)$ for all $z \in \{x, y\}$. Then

a) it holds that $x.\text{wl}(v) = y.\text{wl}(v)$,

b) for all $u \in \text{FUN}(v)$, it holds that $x.\text{wl}(u) = y.\text{wl}(u)$, and

c) for all $r \in \delta^+_G(v)$ where r is not full in x, it holds that $x.f(r) = y.f(r)$.

Proof. We prove the claims individually.

a) As $z.\text{pce}(\text{PS}(v)) < \text{thr}(\text{PS}(v))$ for all $z \in \{x, y\}$, it holds that v is non-flooded in both x and y.

b) Let $u \in \text{FUN}(v)$. If $\text{PS}(u)$ is an ftc sink in x, then it is also in y due to Lemmas 3.78 and 3.79, which means that the claim holds in this case. If $\text{PS}(u)$ is not an ftc sink in x, it is, again because of Lemmas 3.78 and 3.79, not an ftc sink in y. In this case, it holds that $x.\text{excess}(\text{PS}(u)) = y.\text{excess}(\text{PS}(u))$, which shows the claim.

c) Let $r \in \delta^+_G(v)$ where r is not full in x. Then, due to Lemma 3.79, it holds that $x.f(r) = x.\text{nfad}(v) \cdot \text{ratio}(r) = y.\text{nfad}(v) \cdot \text{ratio}(r) = y.f(r)$, which proves the claim.

□

Using this proposition, the desired property of the water levels can be shown.

Proposition 3.81 For all $v \in V$, it holds that $x.\text{wl}(v) = y.\text{wl}(v)$.

Proof. Let $v' \in V$ be the highest node in the graph. As v' is non-flooded due to Assumption (A2), it clearly holds that v' is explored above. If $\text{pce}(\text{PS}(v')) = \text{thr}(\text{PS}(v'))$, we know that all pre-sinks induced by the follow-up nodes of v' are ftc sinks, which means that the claim holds. If this is not the case, Proposition 3.80 shows that, for each $u \in \mathrm{FUN}(v')$, it holds that $x.\mathrm{wl}(u) = y.\mathrm{wl}(u)$. If $\mathrm{wl}(u) > 0$, it must hold that $\mathrm{PS}(u)$ is a sink, and, hence, that all nodes in $\mathrm{PS}(u)$ have the same water level in the two solutions. Otherwise, Proposition 3.80 shows that u is explored above. We then continue applying the same arguments to all such u and work our way down the graph until we have shown the claim for all nodes. □

This proposition completes the proof of the main theorem of this section.

Theorem 3.82 Let $D \subseteq \mathcal{A}$ be a feasible set of actions. For any solution x of the MIP taking exactly the actions in D and the result y of Algorithm 3 applied on G^D, it holds that $x.\mathrm{wl}(v) = y.\mathrm{wl}(v)$ for all $v \in V$.

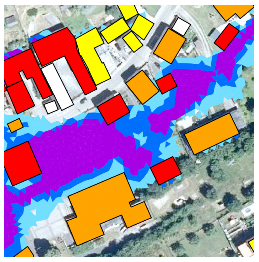
3.5 Computational Results

In this section, we present a comparison of the results obtained from our MIP to results obtained from established simulation software. Afterwards, we use real-world instances from different municipalities to identify and analyze the main drivers for the running time of our method and the quality of the obtained solutions.

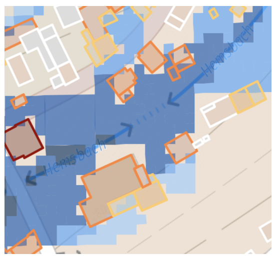
3.5.1 Comparison with Established Simulation Software

To validate our approach, we compare the results obtained on real-world instances to results obtained on these instances from the well-established simulation software “HYSTEM-EXTRAN” [Ins], which is the German industry standard for hydro-dynamic simulations in urban water management and is used by most engineering offices and municipalities when evaluating precautionary measures for pluvial flash floods. It does, however, not support any kind of optimization, but can only be used to simulate the water levels resulting from a given (usually manually chosen) combination of actions for a given amount of rain. Thus, we compare the water levels resulting from our approach for the status quo of each instance (which contains only the already implemented actions, if any) without allowing any additional actions to the water levels obtained from HYSTEM-EXTRAN’s simulation for the same situation. The results obtained from HYSTEM-EXTRAN have been provided and validated by the engineering office igr AG, which was one of our partners in the project AKUT.

All in all, it is found that the results predominantly coincide, with only slight differences that usually occur at the periphery of flooded areas. An illustrative example, in which a 30-year rain event (i.e., the heaviest rain to be expected in the chosen area over a time span of 30 years) is simulated in a hilly region, is provided in Figure 3.7. We further observe that AKUT slightly underestimates the damage to buildings in hilly regions whereas it slightly overestimates the damage to buildings in flat regions. This is due to the fact that HYSTEM-EXTRAN also takes damage caused by high current velocity into account, which is neglected in AKUT.



(a) Water levels obtained from HYSTEM-EXTRAN.



(b) Water levels obtained from the software AKUT.

Figure 3.7.: Extract from a comparison of water levels obtained from HYSTEM-EXTRAN and AKUT for a 30-year rain event in a hilly region. The darker the blue color, the higher the water level, where the highest obtained levels are illustrated in purple in the case of HYSTEM-EXTRAN.

3.5.2 Running Time and Performance

We now investigate the running times of our MIP and the quality of the obtained solutions. For both of them, we present the most important drivers that have been identified by applying our approach to a wide range of different real-world problem instances during the project AKUT. As an illustration, results for nine representative instances obtained from three different regions (two municipalities and a part of a city) that are considered in three relevant scenarios are presented. The three scenarios are a 30-year rain event with a budget that allows to take four actions, a 50-year rain event with with a budget that allows to take four actions, and a 50-year rain event with with a larger budget that allows to take six actions.[19] The regions are a municipality on a hilly terrain, called “Hilly Region” (HR) in the following, a municipality on a flat terrain, called “Flat Region 1” (FR1) in the following, and a part of a city on a flat terrain, called “Flat Region 2” (FR2) in the following. For each region, the set $\mathcal{A}$ of possible actions is the same for all three scenarios and consists of about 20 actions that have been selected according to the local circumstances such that each of them could be implemented in reality.

For each instance, an initial solution taking no actions, which is computed using Algorithm 3, is given to the MIP. As termination criterion, a 3% MIP gap is used for the hilly region, and a 5% MIP gap is used for the flat regions. Further, a time limit of 24 hours is set. All computations in this section were executed using Gurobi 9.5.0 on a server with 32 AMD EPYC 7542 processors (2.9GHz). The most important characteristics of the instances together with the results and the running times are provided in Table 3.3.

In general, it is found that the maximum possible number of actions is taken in each of the nine instances. It is worth noting that there are instances, which are not presented here, where this is not the case. Possible reasons for not taking an action although it would be possible are (1) an overabundance of possible actions and a large budget such that the action that is not taken does not contribute to the quality of the solution anymore and (2) a poor-quality location of the action such that the action does not protect any buildings.

[19]Recall that a 30-year (50-year) rain event corresponds to the heaviest rain to be expected in the chosen area over a time span of 30 years (50 years).

Region	Total Area $[m^2]$	# Buildings	HM
Hilly Region (HR)	2294375	579	7.2%
Flat Region 1 (FR1)	1728799	573	1.9%
Flat Region 2 (FR2)	585123	957	2.0%

Instance	$\lvert V_{\text{red}} \rvert$	Running Time	FSF	IOV	BOV
HR, 30-year, 4 actions	4719	48min	31min	821	807
HR, 50-year, 4 actions	4750	2h 20min	44min	863	855
HR, 50-year, 6 actions	4750	56min	42min	863	854
FR1, 30-year, 4 actions	6613	1h 8min	33min	405	392
FR1, 50-year, 4 actions	6646	24h	4h 26min	418	399
FR1, 50-year, 6 actions	6646	24h	2h 35min	418	398
FR2, 30-year, 4 actions	3778	6h 36min	6h 36min	571	475
FR2, 50-year, 4 actions	3790	3h 2min	2h 33min	575	483
FR2, 50-year, 6 actions	3790	2h 27min	2h 27min	575	480

Table 3.3.: Computational results for nine representative instances. The column "HM" (**H**illiness **M**eassure) contains the median of the values obtained by dividing the slope of each arc by the Euclidean distance of the centers of its incident nodes, which is a measure of how hilly the terrain is. The column "FSF" contains the time until the final solution is found. The column "IOV" contains the objective value of the initial solution of the MIP provided by applying Algorithm 3. The column "BOV" contains the objective value of the best solution returned by the MIP. Note that the different values of $\lvert V_{\text{red}} \rvert$ among instances with the same region result from different merging of nodes during preprocessing due to different rain events.

Among the 42 actions that are selected by the MIP in the presented instances, 40 are basins, while only two are ditches or embankments. This confirms observations made on numerous real-world instances indicating that retention basins, if they can be built, are usually the most efficient actions. However, building retention basins requires free space, which is not always available, especially in densely populated urban areas. Embankments and ditches are usually built together with a retention basin such that the actions overlap geographically. An intuitive explanation for this behavior of the MIP is that retention basins act as a storage for the water, while ditches or embankments connect the inflow from a larger area to the basins.

Another interesting finding is that, in hilly regions, the most efficient retention basins, i.e., the ones that are typically selected by the MIP, are often low-lying and located centrally. As an illustration, in the three scenarios considered for HR, 10 of the 14 selected basins are low-lying and located centrally.

Although the rain volume is significantly higher in the instances modeling the 50-year rain events, it is found that neither the budget nor the rain volume dramatically change the set of taken actions. Among the 12 actions taken in the three considered instances with a 50-year rain event and four possible actions per instance, eight are also taken in the corresponding instances with a 30-year rain event. Further, among the 12 actions taken in the three instances with a 50-year rain event and six possible actions per instance, nine are also taken in the corresponding instances with four possible actions.

In general, the running times show high fluctuations as can be seen in FR1, where the instance with the 30-year rain event takes significantly less time to solve than the instances with the 50-year rain event. Still, several factors influencing the running time and the quality of the obtained solutions can be identified.

Concerning the running time, the most important factor is the number of nodes in the graph (i.e., $|V_{\text{red}}|$). The instances of FR1, which are the instances with the largest number of nodes, with a 50-year rain event are the only instances in our experiments on which the MIP gap could not be closed before reaching the time limit, whereas all other instances could be solved within less than 7 hours.

The second most important factor is the hilliness of the terrain surface. Comparing HR and FR2, the instances for HR are solved significantly faster than the instances for FR2 despite the graph for FR2 being slightly smaller than the graph for HR. Further, in hillier regions, the MIP tends to be numerically more stable.

Although the parameter “MIPFocus” is set to 2 and the parameter “Heuristics” is set to 0.01, which both enforce a stronger attention on improving the lower bound, the final solution is usually found relatively quickly, and the solver spends a significant part of the overall running time on improving the lower bound afterwards, as can be seen when comparing the values in the columns “Running Time” and “FSF” (final solution found) in Table 3.3. Without tuning the parameters accordingly, the running time increases drastically since the solver struggles to close the MIP gap.

Concerning the quality of the solutions, we observe that hillier regions usually have more damage potential overall. To illustrate this, we compare HR to FR1, which have almost the same number of buildings. However, the objective values of both the initial solution and the solution returned by the MIP are more than twice as large in HR as in FR1. This is due to two reasons. Firstly, hilly regions have heavier rainfalls than flat regions due to orographic precipitation [Roe05]. In our example, a 30-year rain event in HR has a precipitation level of 44.9mm whereas a 30-year rain event in FR1 has a precipitation level of only 35.9mm. Secondly, hilly regions tend to have a larger drainage area, which can also be seen comparing the total areas of HR and FR1. Indeed, the difference in the total areas result almost entirely from a higher number of water-dispensing nodes in instances of HR.

Lastly, the density of the buildings (i.e., the number of buildings per area) affects the potential of how much better the solution returned by the MIP can be compared to the initial solution (i.e., the solution where no actions are taken). In our case, there is a significantly higher density of buildings in FR2 than there is in HR and FR1. We see that the difference between the objective values of the initial solution and the obtained solution from the MIP is considerably larger in FR2 than it is in HR and FR1. This stems from the fact that, if a high water level at a critical location is prevented by an action, the action protects more buildings in FR2 than it does in the other two regions. It is worth noting, however, that there are other instances with a high density of buildings where planning impactful actions becomes hard due to lack of space. In such cases, a high density of buildings can decrease the potential for damage reduction by taking actions significantly.

3.6 Conclusion

To the best of our knowledge, the web application AKUT is the first software that uses optimization techniques to support decision making in planning precautionary measures for pluvial flash floods and scales well enough to be applied to realistic scenarios. Since its release, it has been used by over 30 organizations from all over Germany. The usage of optimization techniques in this context has evidently provided valuable support in handling a challenging and highly topical task for various organizations like municipalities, engineering offices, research institutes, and many more.

A mixed-integer program is used to minimize the damage in the case of a heavy rain event by taking best-possible actions subject to a limited budget and constraints on the cooperation of residents. To model the terrain surface, a grid graph obtained from a digital terrain model is transformed by several preprocessing methods such that the cardinality of its node set becomes small enough to apply the previously mentioned mixed-integer program while still maintaining a realistic representation of the terrain surface. Comparisons with results from established software provide strong evidence that solutions obtained from our approach yield realistic results.

When applying the software to large cities, these must currently be subdivided into several parts due to performance reasons. Hence, an interesting question would be how the performance of our approach could further be improved such that it can handle larger instances. As the problem decomposes into several smaller subproblems, using decomposition methods could be a promising attempt. Another possible approach for improving the running times of the model could be to implement callbacks that use Algorithm 3 to compute new solutions during later stages of the branch and bound process.

4 A $(B+1)$-Approximation for Network Flow Interdiction with Unit Removal Costs

Abstract In the network flow interdiction problem (NFI), an interdictor aims to remove arcs of total cost at most a given budget B from a network with given arc costs and capacities such that the value of a maximum flow from a source s to a sink t is minimized. We present a polynomial-time $(B+1)$-approximation algorithm for NFI with unit arc costs, which is the first approximation algorithm for any variant of network flow interdiction whose approximation ratio only depends on the budget available to the interdictor, but not on the size of the network.

4.1 Introduction

While the mixed-integer programming approach presented in the previous chapter influences the flows by modifying the geodesic heights of the nodes, another well-studied approach to modeling decision strategies influencing the flows on a network is to remove arcs from the network. In the context of pluvial flood mitigation, building a ditch could be modeled by removing the arcs from the network that are incident to nodes that intersect with the ditch. While this technique has not been applied in the software AKUT, we though conducted research on the well-known network flow interdiction problem. The majority of the results in this chapter are published in [BT20; BT21].

The class of problems, where an interdictor aims to remove arcs or nodes of total cost at most a given budget from a (directed or undirected) graph or network such that the optimal objective value of an optimization problem on the resulting graph or network is maximized (in case of a minimization problem) or minimized (in case of a maximization problem), is called the class of *interdiction problems*. Due to their high applicability and their immediate connection to assessing the robustness of graphs and networks, interdiction problems have been studied for the vast majority of important graph-related optimization problems. A survey on interdiction problems can be found in [SS20].

In this chapter, the network flow interdiction problem, where an interdictor aims to remove arcs of total cost at most a given budget B from a network with given arc costs and capacities such that the value of a maximum flow from a source s to a sink t is minimized, is studied.

4.1.1 Previous Work

The problem was first stated in 1964 [Wol64] and has been widely studied since then due to its numerous applications ranging from highway transportation [Dur66] over targeting strikes in a lines-of-communication network [Wol70] to critical infrastructure analysis [MMG07]. The problem formulation used in most of the literature today has been introduced in [Phi93], where multiple hardness results on different classes of graphs are shown. Furthermore, a pseudopolynomial-time algorithm and an FPTAS for the problem on planar graphs are presented. A proof of strong $\mathcal{NP}$-hardness of NFI, which also holds true for unit arc costs, can be found in [Woo93]. An algorithm for NFI that, for any $\varepsilon > 0$, either returns a $(1 + {}^{1}\!/\!{\varepsilon})$-approximate solution or a (super-)optimal solution violating the budget by a factor of at most $(1+\varepsilon)$ is presented in [Bur+03]. So far, the best known polynomial-time approximation algorithm for NFI is presented in [CZ17] and achieves an approximation ratio of $2(n-1)$, where n is the number of nodes in the network. They also present a hardness-of-approximation result using a reduction from the densest k subgraph problem, which itself is known to be hard to approximate assuming the exponential time hypothesis [Man17]. Various extensions of the problem involving multiple terminals [ATW11] or multiple objective functions [RW07; Sch+20] have also been studied in the literature.

An interesting related field of research worth mentioning are network improvement problems, where the aim is to improve the network instead of harming it. A variety of network improvement problems, an analysis of their complexity, and approximation algorithms are presented in [Kru+98] and a time-expanded version is investigated in [SL06]. The application of network improvement problems ranges from improving accessibility to rural health services [MC09] to the design of new bike lanes [LFH16].

The approximation algorithm for NFI we present in this chapter dynamically scales the capacities of a subset of arcs in the network and uses an algorithm for the minimum s-t-cut problem as a subroutine. For an overview of state-of-the-art minimum s-t-cut algorithms, we refer to [AMO93]. To the best of our knowledge, the fastest algorithm on directed networks so far, which we also use in our algorithm, with running time in $\mathcal{O}(nm)$ is presented in [Orl13].

4.1.2 Our Contribution

We present a $(B+1)$-approximation algorithm for NFI with unit arc costs. This algorithm extends the algorithm presented in the conference version of this work [BT20], which achieves an approximation ratio of $(B+1)$ only for the special case of unit-cost arcs that may only have a small or a large capacity (i.e., only two different values are allowed for the arc capacities), to the case where arcs still have unit removal costs, but arbitrary capacities. To the best of our knowledge, our algorithm is the first algorithm for any variant of NFI to achieve an approximation ratio that only depends on the interdiction budget B, but not on the size of the network. Moreover, we show that our analysis of the algorithm is essentially tight. Additionally, we demonstrate that the approximation ratio of our algorithm can be improved to $B-k+1$ for any constant $k \leqslant B$ by a boosting argument.

The best approximation algorithm known for NFI with unit arc costs so far is the algorithm of Chestnut and Zenklusen [CZ17], whose approximation ratio improves to $(n-1)$ in the case of unit arc costs. Since, for instances on simple graphs, the source s can always be separated from the sink t for any budget $B \geqslant n-1$ by simply removing the at most $n-1$ arcs starting in s, the approximation ratio we obtain thus dominates the previously best known approximation ratio for NFI with unit arc costs on simple graphs.

4.2 Problem Definition and Structural Results

Let $G = (V, R, u)$ be a directed network, consisting of a node set V, a set R of directed arcs, and a capacity function $u : R \to \mathbb{Q}_{>0}$. Furthermore, let $s \neq t$ be two nodes in G, and let $B \in \mathbb{N}_{>0}$ be an interdiction budget. The network flow interdiction problem with unit costs (which we denote by NFI in the following) asks for a subset $D \subseteq R$, called *interdiction strategy*, of arcs with cardinality $|D| \leqslant B$ such that the value of a maximum s-t-flow in the network $G_D := (V, R\backslash D, u|_{R\backslash D})$ is minimized (where $u|_{R\backslash D}$ denotes the restriction of the capacity function u to $R\backslash D$). Formally, the problem can be stated as follows:

Network Flow Interdiction

INSTANCE:	A directed network $G = (V, R, u)$, two nodes $s, t \in V$ with $s \neq t$, and a budget $B \in \mathbb{N}_{>0}$
TASK:	Find a subset $D \subseteq R$ of arcs with $|D| \leqslant B$ such that the value $\text{val}^G(D)$ of a maximum s-t-flow in the network $G_D = (V, R\backslash D, u|_{R\backslash D})$ is minimized.

For an instance of NFI, the number of different capacities is bounded from above by the number $m = |R|$ of arcs in the network. Thus, we can assume, that $u : R \to \{u_1, \ldots, u_k\}$ for some $k \leqslant m$ and $u_1 < \cdots < u_k$. Furthermore, the smallest arc capacity u_1 can be assumed to be one by simply dividing all capacities by u_1 if $u_1 \neq 1$. For ease of notation, we define $\bar{u} := u_k$. Lastly, we assume that the arcs $r_1, \ldots, r_m$ in the set R are numbered by non-decreasing capacities, breaking ties arbitrarily. In other words, for two arcs $r_i, r_j \in R$ with $i < j$, it holds that $u(r_i) \leqslant u(r_j)$. Note that, if this is not the case, this sorting can be performed in $\mathcal{O}(m \log(m))$ time.

Whenever we refer to the network $G = (V, R, u)$ with capacity function $u : R \to \mathbb{Q}_{>0}$, we mean the input network of the instance. Throughout this chapter, we also consider slight modifications of the network G, where the node and arc sets remain unchanged, but the capacity function is changed. Whenever we refer to such a modification, we denote the network by $H = (V, R, u_H)$ with capacity function $u_H : R \to \mathbb{Q}_{>0}$. In any such network H, we adopt the numbering of the arcs from G. Note that this numbering might not be in non-decreasing order of capacities with respect to the capacity function u_H.

We call an instance of NFI *trivial* if its optimum objective value equals zero, and *non-trivial*, otherwise. By the well-known max-flow min-cut theorem (see [FF62]), it is easy to check in polynomial time whether a given instance is trivial by testing whether, when setting all arc capacities to one, a minimum cut in G has capacity at most B. In the following, we assume that all instances are non-trivial.

4.2.1 Structural Results

When solving the network flow interdiction problem, s-t-cuts play a central role. In this section, we show how the concept of removing arcs can be translated to a concept of attacking a cut. The first observation follows directly from the max-flow min-cut theorem:

Observation 4.2 For any interdiction strategy $D \subseteq R$, its objective value $\mathrm{val}^G(D)$ for NFI equals the capacity of a minimum s-t-cut in the interdicted network G_D.

The computation of minimum cuts plays an important role in our algorithm and its analysis. Throughout the chapter, we assume the use of an arbitrary but fixed (deterministic) algorithm to compute a minimum s-t-cut in a given network in polynomial time.[1] For a solution D, we denote the minimum cut in the interdicted network G_D computed by this minimum cut algorithm by $C_D = (S_D, T_D)$. The cut C_D is interpreted as a cut in the original network $G = (V, R, u)$. The next lemma uses an exchange argument and motivates the investigation of cuts when finding interdiction strategies.

Lemma 4.3 For an interdiction strategy $D \subseteq R$ and any minimum cut (S, T) in the interdicted network G_D, it either holds that $D \subseteq \delta^+_G(S)$, or $\mathrm{val}^G(D)$ can be reduced by removing an arc in $D \backslash \delta^+_G(S)$ from D and adding an arc from $\delta^+_G(S) \backslash D$ to D.

Proof. Assume that there exists an arc $r \in D \backslash \delta^+_G(S)$. Since the instance is non-trivial, there must also exist an arc $r' \in \delta^+_G(S) \backslash D$. Now let $D' := (D \backslash \{r\}) \cup \{r'\}$. By Observation 4.2, removing r from the solution D does not change its value $\mathrm{val}^G(D)$, but adding r' to D decreases $\mathrm{val}^G(D)$ by $u(r') > 0$. Therefore, it holds that $\mathrm{val}^G(D') = \mathrm{val}^G(D) - u(r') < \mathrm{val}^G(D)$. $\square$

The lemma immediately implies an important property of optimal interdiction strategies.

Corollary 4.4 For an optimal solution D^{OPT}, any minimum cut (S, T) in the interdicted network $G_{D^{\mathrm{OPT}}}$ satisfies $D^{\mathrm{OPT}} \subseteq \delta^+_G(S)$.

While any interdiction strategy D is identified with a cut C_D, a given cut C can also be identified with an interdiction strategy by using the interdiction budget B to reduce the capacity of the cut in G as far as possible, which can be easily achieved by removing the B arcs of largest index, i.e., the B arcs with largest capacity, in C. This motivates the following definition:

Definition 4.5 Let $H = (V, R, u_H)$ be a network with the same node and arc set as G. For a cut C in H, we define the strategy of *attacking the cut* as the solution $D_C \subseteq R$ containing the B arcs of largest index in the cut C.[2] Interdiction strategies of the form D_C are called

[1]For an overview of minimum cut algorithms, we refer to [AMO93].

[2]Note that all networks that we consider throughout the chapter are non-trivial. Hence, any cut must contain at least $B + 1$ arcs.

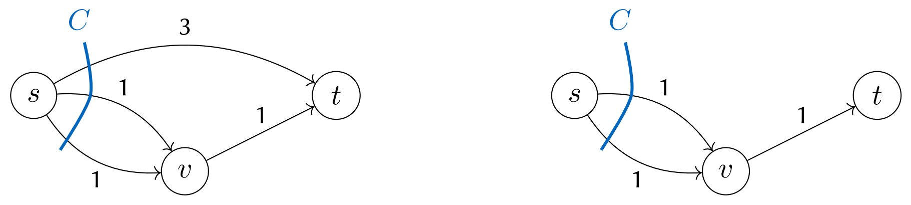


Figure 4.1.: The network G of the instance described in Example 4.6 on the left hand side and the network G_{D_C} on the right hand side. The cut C is optimal, while it is not minimum in G_{D_C}.

cut interdiction strategy. The smallest among the indices of the arcs in the cut interdiction strategy D_C is called the *lowest removal index* of the cut C. Furthermore, we define the *value of a cut* C in H as $\mathrm{val}^H(C) := \mathrm{val}^H(D_C)$ and call a cut C *optimal on* H if D_C is an optimal interdiction strategy for NFI on H. Lastly, we define the *interdicted capacity* of a cut C in H as $\mathrm{icap}^H(C) := \mathrm{cap}^{H_D}(C) = \mathrm{cap}^H(C) - u_H(D_C)$, i.e., the capacity of the cut after interdiction.

For simplicity, if a cut C is optimal for NFI on G, we just say that C is optimal without explicitly referring to the network G. Note that, even though the notion of attacking a cut C is defined for any modified network $H = (V, R, u_H)$, for later convenience, the arcs in the interdiction strategy D_C are chosen in a way that reduces the capacity of the cut in G (and not necessarily in H) as far as possible. In particular, the cut interdiction strategy D_C is independent of which network $H = (V, R, u_H)$ is considered. Moreover, note that removing arcs from a cut C in the described way does not imply that C is a minimum cut in the interdicted networks H_{D_C} or G_{D_C}, not even if C is optimal for NFI on H or G. This is corroborated in the following example.

Example 4.6 Let $G = (V, R, u)$ be a network with $V = \{s, v, t\}$, and the arc set consisting of one arc from s to t with capacity 3, two parallel arcs from s to v with capacity 1 and one arc from v to t with capacity 1. Further, let $B = 1$. The cut $C := (\{s\}, \{v, t\})$ is optimal, but it is not a minimum cut in the interdicted network G_{D_C}. This is illustrated in Figure 4.1.

Example 4.6 motivates the following definition:

Definition 4.7 We call a cut C *minimum after interdiction in a network* $H = (V, R, u_H)$ if it is a minimum cut in the interdicted network H_{D_C}. If the network is clear from the context, we only say a cut is minimum after interdiction without explicitly referring to the network.

The next lemma follows from the fact that the capacity of a cut is an upper bound on the value of a maximum s-t-flow in a network and from the max-flow min-cut theorem (see [AMO93]):

Lemma 4.8 Let $H = (V, R, u_H)$ be a network and let C be a cut in H. Then it holds that $\mathrm{val}^H(C) \leqslant \mathrm{icap}^H(C)$. This holds with equality if and only if C is minimum after interdiction in H.

Proof. The interdicted capacity $\mathrm{icap}^H(C)$ is defined as the capacity of C in H_{D_C}, whereas the value $\mathrm{val}^H(C)$ is defined as the value of a maximum s-t-flow in H_{D_C}, which, due to the max-flow min-cut theorem, equals the capacity of a minimum cut in H_{D_C}. Therefore, it holds that $\mathrm{val}^H(C) \leqslant \mathrm{icap}^H(C)$. Now, C is minimum after interdiction in H if it is a minimum cut in H_{D_C}, which, with the above argumentation, holds if and only if $\mathrm{val}^H(C) = \mathrm{icap}^H(C)$. □

Being minimum after interdiction is a desirable property of cuts – in particular of optimal cuts. This motivates the following lemma:

Lemma 4.9 There exists an optimal cut C^{OPT} that is minimum after interdiction in G.

Proof. Let D^{OPT} be an optimal interdiction strategy for NFI on G and let C^{OPT} be a minimum cut in $G_{D^{\mathrm{OPT}}}$. Due to Corollary 4.4, the cut C^{OPT} must contain all arcs of D^{OPT}. Then, it must hold that $u(D_{C^{\mathrm{OPT}}}) \leqslant u(D^{\mathrm{OPT}})$ since, otherwise, $D_{C^{\mathrm{OPT}}}$ would remove strictly more capacity from the cut C^{OPT} than D^{OPT}. As C^{OPT} is a minimum cut in $G_{D^{\mathrm{OPT}}}$, this would lead to a strictly better interdiction strategy, contradicting the optimality of D^{OPT}. Conversely, it holds that $u(D_{C^{\mathrm{OPT}}}) \geqslant u(D^{\mathrm{OPT}})$ as $D_{C^{\mathrm{OPT}}}$ contains the B arcs of largest index and, hence, of maximum capacity in C^{OPT} and all the at most B arcs in D^{OPT} are contained in C^{OPT}. Thus, it holds that $u(D_{C^{\mathrm{OPT}}}) = u(D^{\mathrm{OPT}})$. This yields

$$\begin{aligned} \mathrm{val}^G(D_{C^{\mathrm{OPT}}}) &\overset{(1)}{\leqslant} \mathrm{icap}^G(C^{\mathrm{OPT}}) = \mathrm{cap}^G(C^{\mathrm{OPT}}) - u(D_{C^{\mathrm{OPT}}}) \\ &= \mathrm{cap}^G(C^{\mathrm{OPT}}) - u(D^{\mathrm{OPT}}) \overset{(2)}{=} \mathrm{val}^G(D^{\mathrm{OPT}}), \end{aligned}$$

where (1) holds due to Lemma 4.8, and (2) follows from C^{OPT} being a minimum cut in $G_{D^{\mathrm{OPT}}}$. Hence, $D_{C^{\mathrm{OPT}}}$ must be an optimal solution and equality holds in (1), which shows that C^{OPT} is also minimum after interdiction in G due to Lemma 4.8. □

For ease of notation, we formally define the notion of a cut being α-approximate.

Definition 4.10 Let $\alpha \geqslant 1$ and let $H = (V, R, u_H)$ be a network. A cut C in H is called *α-approximate* for NFI on H if D_C is α-approximate for NFI on H.

Clearly, if an (approximately) optimal cut C is known, the corresponding (approximately) optimal interdiction strategy D_C can easily be computed in polynomial time. Therefore, the challenge in approximating NFI on a network G lies in finding a cut C whose value $\mathrm{val}^G(C)$ is as low as possible. Next, we define a network in which the capacities of arcs with large indices are scaled. This network plays a central role in our algorithm.

Definition 4.11 Let $\gamma \geqslant 1$ and $l \in \mathbb{N}$ with $l \leqslant m$. We define the capacity function $u_{l,\gamma} : R \to \mathbb{Q}_{>0}$ by

$$u_{l,\gamma}(r_i) := \begin{cases} \gamma, & \text{if } i \geqslant l \\ u(r_i), & \text{else,} \end{cases}$$

i.e., all arcs of index greater or equal to l have capacity γ while the capacity of the other arcs remains unchanged. Furthermore, we set $G(l,\gamma) := (V, R, u_{l,\gamma})$ and define $C(l,\gamma)$ as the minimum cut in $G(l,\gamma)$ returned by the deterministic minimum cut algorithm.

Recall that $\bar{u}$ is defined as the largest capacity in G. The intuition behind the network $G(l,\gamma)$ is that any cut C in the network G has a larger value in $G(l,\bar{u})$ than it has in G, and an optimal cut C_{OPT} has the same value for NFI on both $G(l,\bar{u})$ and G if l is larger than or equal to the cut's lowest removal index. This is summarized in the following lemma:

Lemma 4.12 Let C be a cut in G. Then, for any $1 \leqslant l \leqslant m$, it holds that $\mathrm{val}^G(C) \leqslant \mathrm{val}^{G(l,\bar{u})}(C)$. Let $\bar{l}$ be the lowest removal index of C. If, additionally, C is minimum after interdiction in G and $l \geqslant \bar{l}$, then C is minimum after interdiction in $G(l,\bar{u})$ and $\mathrm{val}^G(C) = \mathrm{val}^{G(l,\bar{u})}(C)$.

Proof. The networks G and $G(l,\bar{u})$ have the same sets of nodes and arcs, and for each of the arcs, it holds that its capacity in G is at most its capacity in $G(l,\bar{u})$. This yields $\mathrm{val}^G(C) \leqslant \mathrm{val}^{G(l,\bar{u})}(C)$. If additionally $l \geqslant \bar{l}$, all arcs with index greater or equal to l are removed from C when attacking C in $G(l,\bar{u})$, which means the remaining arcs in C after interdiction have the same capacities in both networks G and $G(l,\bar{u})$. In other words, this means that $\mathrm{icap}^G(C) = \mathrm{icap}^{G(l,\bar{u})}(C)$. Further, if C is minimum after interdiction in G, it holds that

$$\mathrm{icap}^G(C) \overset{(1)}{=} \mathrm{val}^G(C) \overset{(2)}{\leqslant} \mathrm{val}^{G(l,\bar{u})}(C) \overset{(3)}{\leqslant} \mathrm{icap}^{G(l,\bar{u})}(C),$$

where (1) and (3) hold because of Lemma 4.8, and (2) has already been shown above. Since it holds that $\mathrm{icap}^G(C) = \mathrm{icap}^{G(l,\bar{u})}(C)$, equality must hold in (2) and (3), which means $\mathrm{val}^G(C) = \mathrm{val}^{G(l,\bar{u})}(C)$ and C is minimum after interdiction in $G(l,\bar{u})$ due to Lemma 4.8. □

Applying Lemma 4.12 to an optimal cut yields the following important result.

Corollary 4.13 Let C^{OPT} be a cut that is optimal and minimum after interdiction in G and let l^{OPT} be its lowest removal index. Then C^{OPT} is also optimal on $G(l^{\mathrm{OPT}},\bar{u})$ and minimum after interdiction in $G(l^{\mathrm{OPT}},\bar{u})$.

For the remainder of this chapter, we denote by $C^\star$ an arbitrary but fixed optimal cut for NFI on G that is minimum after interdiction, and by $l^\star$ its lowest removal index. Note that such a cut exists due to Lemma 4.9. Due to Corollary 4.13, the cut $C^\star$ is also optimal and minimum after interdiction in $G(l^\star,\bar{u})$.

The following lemma shows that an α-approximate cut for NFI on $G(l^\star,\bar{u})$ is also α-approximate for NFI on G.

Lemma 4.14 Any α-approximate cut C for NFI on $G(l^\star,\bar{u})$, is also an α-approximate cut for NFI on G.

Proof. It holds that

$$\mathrm{val}^G(C) \overset{\text{Lem. 4.12}}{\leqslant} \mathrm{val}^{G(l^\star,\bar{u})}(C) \leqslant \alpha \cdot \mathrm{val}^{G(l^\star,\bar{u})}(C^\star) \overset{\text{Lem. 4.12}}{=} \alpha \cdot \mathrm{val}^G(C^\star).$$ □

4.3 A $(\mathrm{B}+1)$-Approximation for NFI

Lemma 4.14 shows that, if we manage to find an α-approximate interdiction strategy for NFI on $G(l^\star, \bar{u})$, we immediately obtain an α-approximate interdiction strategy for NFI on G. A problem of this procedure is that $l^\star$ is unknown and cannot be computed efficiently. However, as $l^\star$ can only attain integer values from 1 to $m - B + 1$, it is possible to iteratively „guess" $l^\star$ and compute an approximate interdiction strategy in each iteration. Out of these $m - B + 1$ interdiction strategies computed for each possible value for $l^\star$, we then take the one with lowest value for NFI on G, which then yields an α-approximate interdiction strategy. This idea is formally summarized in the following lemma:

Lemma 4.15 Let $\alpha \geqslant 1$. If there exists an algorithm ALG for NFI with additional input $l \in \{1, \ldots, m - B + 1\}$ that

1) terminates in $\mathcal{O}(\mathcal{T})$ time for any $l \in \{1, \ldots, m - B + 1\}$ and

2) returns an α-approximate cut interdiction strategy for NFI on $G(l^\star, \bar{u})$ for $l = l^\star$,

then applying ALG for $l = 1, \ldots, m - B + 1$ and returning an interdiction strategy with lowest value for NFI on G yields an α-approximation algorithm for NFI on G that runs in $\mathcal{O}(m \cdot \mathcal{T})$ time.

Proof. Let $D_1, \ldots, D_{m-B+1}$ be the interdiction strategies returned by ALG applied for $l = 1, \ldots, m - B + 1$, respectively and let $D_{\bar{l}}$ be the one returned by the described procedure, i.e., the one with lowest value. Further, let $D_{C^{l^\star}}$ be the cut interdiction strategy returned by ALG for $l = l^\star$. Then it holds that

$$\mathrm{val}^G(D_{\bar{l}}) \leqslant \mathrm{val}^G(D_{C^{l^\star}}) = \mathrm{val}^G(C_{l^\star}) \leqslant \alpha \cdot \mathrm{val}^G(C^\star)$$

where the last inequality follows by Lemma 4.14.

The statement for the running time follows as ALG runs in $\mathcal{O}(\mathcal{T})$ time and has to be executed $\mathcal{O}(m)$ times, which yields a total running time of $\mathcal{O}(m \cdot \mathcal{T})$. Identifying a solution with lowest value for NFI on G among $D_1, \ldots, D_{m-B+1}$ is possible in $\mathcal{O}(m \cdot \tilde{\mathcal{T}})$ time, where $\tilde{\mathcal{T}}$ denotes the maximum time needed to compute a maximum s-t-flow in one of the networks G_{D_l}, for $l = 1, \ldots, m - B + 1$.[3] □

The scalable capacity function $u_{l,\gamma}$ treats arcs with index larger or equal to a given index l differently than the other arcs, which motivates the following definition:

Definition 4.16 Let $l \in \{1, \ldots, m - B + 1\}$ and let $\gamma \geqslant 1$. All arcs in $G(l, \gamma)$ with index larger than or equal to l are called *large arcs*, and all other arcs in $G(l, \gamma)$ are called *small arcs*. For a cut C in $G(l, \gamma)$, we denote by $\mathrm{nla}^l(C)$ the number of large arcs in C and by $\mathrm{csa}^l(C) := \sum_{\substack{r_i \in C:\\ i<l}} u(r_i)$ the sum of the capacities of the small arcs in C.[4]

[3] We assume that $\tilde{\mathcal{T}} \in \mathcal{O}(\mathcal{T})$.

[4] nla stands for „**n**umber of **l**arge **a**rcs" and csa stands for „**c**apacity of **s**mall **a**rcs".

Before stating the algorithm, we present one more structural result about the number of large arcs in cuts of the form $C(l,\gamma)$[5] for $l \in \{1,\dots,m-B+1\}$ and $\gamma \geqslant 1$. The lemma states that, for fixed l, the number of large arcs in $C(l,\gamma)$ (weakly) increases if γ is decreased.

Lemma 4.17 Let $l \in \{1,\dots,m-B+1\}$ and $1 \leqslant \gamma_1 < \gamma_2 \leqslant \bar{u}$. Then it holds that $\text{nla}^l(C(l,\gamma_1)) \geqslant \text{nla}^l(C(l,\gamma_2))$.

Proof. Let $\text{nla}_1 := \text{nla}^l(C(l,\gamma_1))$, $\text{csa}_1 := \text{csa}^l(C(l,\gamma_1))$ and $\text{nla}_2 := \text{nla}^l(C(l,\gamma_2))$, $\text{csa}_2 := \text{csa}^l(C(l,\gamma_2))$. By definition of the two cuts, we have

$$\begin{aligned}\text{cap}^{G(l,\gamma_1)}(C(l,\gamma_1)) &\leqslant \text{cap}^{G(l,\gamma_1)}(C(l,\gamma_2))\\ \text{cap}^{G(l,\gamma_2)}(C(l,\gamma_2)) &\leqslant \text{cap}^{G(l,\gamma_2)}(C(l,\gamma_1)),\end{aligned}$$

which yields

$$\text{nla}_1 \cdot \gamma_1 + \text{csa}_1 = \text{cap}^{G(l,\gamma_1)}(C(l,\gamma_1)) \leqslant \text{cap}^{G(l,\gamma_1)}(C(l,\gamma_2)) = \text{nla}_2 \cdot \gamma_1 + \text{csa}_2 \tag{4.1}$$

$$\text{and } \text{nla}_2 \cdot \gamma_2 + \text{csa}_2 = \text{cap}^{G(l,\gamma_2)}(C(l,\gamma_2)) \leqslant \text{cap}^{G(l,\gamma_2)}(C(l,\gamma_1)) = \text{nla}_1 \cdot \gamma_2 + \text{csa}_1. \tag{4.2}$$

Adding (4.1) and (4.2) yields

$$\text{nla}_1 \cdot \gamma_1 + \text{csa}_1 + \text{nla}_2 \cdot \gamma_2 + \text{csa}_2 \leqslant \text{nla}_2 \cdot \gamma_1 + \text{csa}_2 + \text{nla}_1 \cdot \gamma_2 + \text{csa}_1.$$

This is equivalent to

$$\text{nla}_2 \cdot (\gamma_2 - \gamma_1) \leqslant \text{nla}_1 \cdot (\gamma_2 - \gamma_1),$$

which means that $\text{nla}_1 \geqslant \text{nla}_2$ since $\gamma_1 < \gamma_2$. $\square$

We now describe our algorithm, whose pseudocode can be found in Algorithm 8. The algorithm takes an instance of NFI and an index $l \in \{1,\dots,m-B+1\}$ as input and uses the recursive bisection procedure stated in Algorithm 9 as a subroutine in order to compute minimum cuts in graphs of the form $G(l,\gamma)$ for some $\gamma \geqslant 1$. Whenever two cuts C_1 and C_2 have been found in the subroutine for γ_1 and γ_2, respectively, the next candidate value $\hat{\gamma}$ is chosen as the value for which the capacities of C_1 and C_2 in $G(l,\hat{\gamma})$ are equal, and the cut $\hat{C} := C(l,\hat{\gamma})$ is computed. If the number of large arcs in $\hat{C}$ is different from both that in C_1 and that in C_2, the bisection method is called recursively for γ_1 and $\hat{\gamma}$, and for $\hat{\gamma}$ and γ_2. Otherwise, the recursion ends and the cuts C_1 and C_2 are returned.

For better readability, we refer to a call to Algorithm 9 by `bisection`(l,γ_1,γ_2) and to the set returned by the algorithm by $\text{bisection}(l,\gamma_1,\gamma_2)$.

The idea of the analysis of the algorithm is to apply Lemma 4.15 to the algorithm. We start by showing that Algorithm 8 terminates in polynomial time for any given index $l \in \{1,\dots,m-B+1\}$, which is the first of the two prerequisites required in order to apply the lemma.

[5]Recall: $C(l,\gamma)$ is the min-cut in $G(l,\gamma)$ returned by the deterministic min-cut algorithm.

Algorithm 8: BISECTION-CUT

```
1 Procedure bisec-cut($G, B, l$)
2     Compute $C(l, 1)$ and $C(l, \bar{u})$
3     if $\mathrm{nla}^l(C(l, 1)) \leqslant B$ then
4         return $D_{C(l,1)}$
5     else if $\mathrm{nla}^l(C(l, \bar{u})) \geqslant B$ then
6         return $D_{C(l,\bar{u})}$
7     else
8         return a cut interdiction strategy $D_C$, where $C \in \underset{\bar{C} \in \text{bisection}(l,1,\bar{u})}{\operatorname{argmin}} \mathrm{val}^G(\bar{C})$
```

Algorithm 9: BISECTION-PROCEDURE

```
1 Procedure bisection($l, \gamma_1, \gamma_2$)
2     $C_1 := C(l, \gamma_1)$
3     $C_2 := C(l, \gamma_2)$
4     Let $\hat{\gamma} := \frac{\mathrm{csa}^l(C_2) - \mathrm{csa}^l(C_1)}{\mathrm{nla}^l(C_1) - \mathrm{nla}^l(C_2)}$ and compute $\hat{C} := C(l, \hat{\gamma})$
5     if $\mathrm{nla}^l(\hat{C}) \notin \{\mathrm{nla}^l(C_1), \mathrm{nla}^l(C_2)\}$ then
6         return bisection ($l, \gamma_1, \hat{\gamma}$) $\cup$ bisection ($l, \hat{\gamma}, \gamma_2$)
7     else
8         return $\{C_1, C_2\}$
```

Observation 4.18 If Algorithm 8 enters one of the if-statements in line 3 or line 5, the whole algorithm only needs to compute two minimum cuts, in which case it runs in $\mathcal{O}(\mathcal{T}_{\mathrm{MC}}(G(l, 1)) + \mathcal{T}_{\mathrm{MC}}(G(l, \bar{u})))$ time, where $\mathcal{T}_{\mathrm{MC}}(H)$ is the time needed to execute the deterministic minimum cut algorithm on a network H.

Due to Observation 4.18, it remains to analyze the running time of Algorithm 8 if it enters the else-statement in line 7. To this end, we first show that Algorithm 9 is indeed a bisection procedure, i.e., for a call to bisection (l, γ_1, γ_2), the new candidate value $\hat{\gamma}$ lies in the interval $[\gamma_1, \gamma_2]$.

Lemma 4.19 Let $l \in \{1, \dots, m - B + 1\}$ be an index for which bisec-cut(G, B, l) enters the else-statement in line 7. Further, let $\gamma_1 \neq \gamma_2$ be two values for which bisection(l, γ_1, γ_2) is called during the execution of Algorithm 8, let C_1 and C_2 be the corresponding cuts from lines 2 and 3, and $\hat{\gamma}$ the value computed in line 4 of Algorithm 9. Then it holds that $1 \leqslant \gamma_1 \leqslant \hat{\gamma} \leqslant \gamma_2 \leqslant \bar{u}$.

Proof. We show by induction over the recursion tree produced by the recursive calls of the bisection procedure that, whenever bisection(l, γ_1, γ_2) is called for two values γ_1, γ_2 as in the claim, then either $1 \leqslant \gamma_1 < \hat{\gamma} < \gamma_2 \leqslant u$, or $\hat{\gamma} \in \{\gamma_1, \gamma_2\}$ and no further recursive calls of the bisection procedure are made within bisection(l, γ_1, γ_2).

We prove the basis of our induction by showing the claim for the root node of the recursion tree, which is the call to bisection$(l,1,\bar{u})$. In this iteration of the recursion, we have $\hat{\gamma} = \frac{\text{csa}^l(C(l,\bar{u}))-\text{csa}^l(C(l,1))}{\text{nla}^l(C(l,1))-\text{nla}^l(C(l,\bar{u}))}$. We first show that $1 \leqslant \hat{\gamma} \leqslant \bar{u}$. For the sake of a contradiction, first suppose that $\hat{\gamma} < 1$. By the definition of $\hat{\gamma}$, it holds that

$$\hat{\gamma} \cdot \text{nla}^l(C(l,1)) + \text{csa}^l(C(l,1)) = \hat{\gamma} \cdot \text{nla}^l(C(l,\bar{u})) + \text{csa}^l(C(l,\bar{u})).$$

As the else-statement in line 7 of Algorithm 8 is entered, it holds that $\text{nla}^l(C(l,1)) > B$ and that $\text{nla}^l(C(l,\bar{u})) < B$. In particular, it holds that $\text{nla}^l(C(l,1)) > \text{nla}^l(C(l,\bar{u}))$, which yields

$$\begin{aligned} 1 \cdot \text{nla}^l(C(l,1)) + \text{csa}^l(C(l,1)) &= \hat{\gamma} \cdot \text{nla}^l(C(l,1)) + \text{csa}^l(C(l,1)) + (1-\hat{\gamma}) \cdot \text{nla}^l(C(l,1)) \\ &> \hat{\gamma} \cdot \text{nla}^l(C(l,\bar{u})) + \text{csa}^l(C(l,\bar{u})) + (1-\hat{\gamma}) \cdot \text{nla}^l(C(l,\bar{u})) \\ &= 1 \cdot \text{nla}^l(C(l,\bar{u})) + \text{csa}^l(C(l,\bar{u})), \end{aligned}$$

implying that $C(l,1)$ is not a minimum cut in $G(l,1)$. This is a contradiction to the definition of $C(l,1)$, and it follows that $\hat{\gamma} \geqslant 1$. Along the same lines, one can prove that $\hat{\gamma} \leqslant \bar{u}$. Consequently, we obtain that $1 \leqslant \hat{\gamma} \leqslant \bar{u}$.

For proving the induction base, it remains to show that, if $\hat{\gamma} \in \{1,\bar{u}\}$, no further recursive calls to the bisection procedure are made within bisection$(l,1,\bar{u})$. First assume that $\hat{\gamma} = 1$. Since the algorithm used for computing the cut $C(l,\hat{\gamma})$ (i.e., a minimum cut in $G(l,\hat{\gamma})$) in line 4 of the bisection procedure is the same determinisctic algorithm that is used to compute $C(l,1)$ in line 2, it then follows that $\hat{C} = C(l,1)$ and, in particular, $\text{nla}^l(\hat{C}) = \text{nla}^l(C(l,1))$. If $\hat{\gamma} = \bar{u}$, it follows along the same lines that $\text{nla}^l(\hat{C}) = \text{nla}^l(C(l,\bar{u}))$. In both cases, this implies that no further recursion steps are made within bisection$(l,1,\bar{u})$, which completes the proof of the induction base.

For the induction step, let bisection(l,γ_1,γ_2) be called during the algorithm and assume that the statement holds for all predecessors in the recursion tree. In particular, this holds for the parent in the recursion tree. Let bisection(l,γ_1',γ_2') be the parent in the recursion tree. Due to the recursive calls in line 6 of Algorithm 9, it must either hold that $\gamma_1' = \gamma_1$ or $\gamma_2' = \gamma_2$. We present the proof for the case that $\gamma_2' = \gamma_2$. The proof in the other case is along the same lines.

Again, due to the structure of the recursive calls in line 6 of the bisection procedure, it must hold that γ_1 and $C(l,\gamma_1)$ are the new candidate value and the cut computed in line 4 of the algorithm in the parent recursion step, respectively. Hence, applying the induction hypothesis for the parent recursion step implies that $1 \leqslant \gamma_1 < \gamma_2 \leqslant u$.

Now consider the call to bisection(l,γ_1,γ_2), in which the cuts $C_1 = C(l,\gamma_1)$, $C_2 = C(l,\gamma_2)$, and $C(l,\hat{\gamma})$ together with the value $\hat{\gamma}$ are computed in lines 2, 3, and 4, respectively. Since $1 \leqslant \gamma_1 < \gamma_2 \leqslant \bar{u}$, Lemma 4.17 shows that $\text{nla}^l(C_1) \geqslant \text{nla}^l(C_2)$. As bisection$(l,\gamma_1,\gamma_2)$ has been called from the parent recursion step, it must also hold that $\text{nla}^l(C_1) \neq \text{nla}^l(C_2)$ by the if-statement in line 5 in the parent recursion. Together, this yields $\text{nla}^l(C_1) > \text{nla}^l(C_2)$.

We show that $\hat{\gamma} \in [\gamma_1,\gamma_2]$. For the sake of a contradiction, first suppose that $\hat{\gamma} < \gamma_1$. By construction, it holds that $\hat{\gamma} \cdot \text{nla}^l(C_1) + \text{csa}^l(C_1) = \hat{\gamma} \cdot \text{nla}^l(C_2) + \text{csa}^l(C_2)$. As $\text{nla}^l(C_1) > \text{nla}^l(C_2)$, it holds that

$$\begin{aligned}\gamma_1 \cdot \mathrm{nla}^l(C_1) + \mathrm{csa}^l(C_1) &= \hat{\gamma} \cdot \mathrm{nla}^l(C_1) + \mathrm{csa}^l(C_1) + (\gamma_1 - \hat{\gamma}) \cdot \mathrm{nla}^l(C_1)\\ &> \hat{\gamma} \cdot \mathrm{nla}^l(C_2) + \mathrm{csa}^l(C_2) + (\gamma_1 - \hat{\gamma}) \cdot \mathrm{nla}^l(C_2)\\ &= \gamma_1 \cdot \mathrm{nla}^l(C_2) + \mathrm{csa}^l(C_2),\end{aligned}$$

which means that C_1 is not a minimum cut in $G(l, \gamma_1)$. This is a contradiction to the choice of C_1, so we obtain that $\hat{\gamma} \geqslant \gamma_1$. Along the same lines, one can prove that $\hat{\gamma} \leqslant \gamma_2$. Consequently, we obtain that $1 \leqslant \gamma_1 \leqslant \hat{\gamma} \leqslant \gamma_2 \leqslant \bar{u}$.

It remains to show that, if $\hat{\gamma} \in \{\gamma_1, \gamma_2\}$, then no further recursive calls of the bisection procedure are made within $\mathtt{bisection}(l, \gamma_1, \gamma_2)$. First assume that $\hat{\gamma} = \gamma_1$. As in the proof of the induction base, it follows that $\hat{C} = C(l, \hat{\gamma}) = C_1$ since the min-cut algorithm is deterministic. In particular, it follows that $\mathrm{nla}^l(\hat{C}) = \mathrm{nla}^l(C_1)$. If $\hat{\gamma} = \gamma_2$, it holds that $\mathrm{nla}^l(\hat{C}) = \mathrm{nla}^l(C_2)$ by the same arguments. In both cases, this implies that the if-statement in line 5 is not entered and, hence, no further recursion steps are made within $\mathtt{bisection}(l, \gamma_1, \gamma_2)$, which completes the proof. □

Using Lemmas 4.17 and 4.19, it is next shown that Algorithm 8 runs in polynomial time.

Proposition 4.20 Algorithm 8 runs in $\mathcal{O}(m \cdot \mathcal{T}_{\mathrm{MC}})$ time, where $\mathcal{T}_{\mathrm{MC}}$ is the maximum time needed to compute a minimum cut in a network of the form $G(l, \gamma)$ with $l \in \{1, \dots, m-B+1\}$ and $\gamma \in [1, \bar{u}]$.

Proof. Observation 4.18 shows that, if Algorithm 8 enters one of the if-statements in line 3 or line 5, the algorithm runs in $\mathcal{O}(\mathcal{T}_{\mathrm{MC}}(G(l,1)) + \mathcal{T}_{\mathrm{MC}}(G(l,\bar{u}))) \subseteq \mathcal{O}(\mathcal{T}_{\mathrm{MC}})$ time. Hence, it remains to analyze the running time of the algorithm if it enters the else-statement in line 7.

A single recursion loop of the bisection procedure can be performed in $\mathcal{O}(\mathcal{T}_{\mathrm{MC}})$ time since the time needed to compute a minimum cut dominates the other steps in a single execution of the procedure. Moreover, whenever $\mathtt{bisection}(l, \gamma_1, \gamma_2)$ is called, Lemma 4.19 shows that $1 \leqslant \gamma_1 \leqslant \hat{\gamma} \leqslant \gamma_2 \leqslant \bar{u}$ for $\hat{\gamma}$ as in line 4 of the procedure. If $\hat{\gamma} \in \{\gamma_1, \gamma_2\}$, the computed cut $\hat{C}$ equals C_1 or C_2 since the algorithm used for computing a minimum cut in lines 2, 3, and 4 of the bisection procedure is deterministic. If $\hat{\gamma} \notin \{\gamma_1, \gamma_2\}$, it holds that $1 \leqslant \gamma_1 < \hat{\gamma} < \gamma_2 \leqslant \bar{u}$, so Lemma 4.17 yields $\mathrm{nla}^l(C_2) \leqslant \mathrm{nla}^l(\hat{C}) \leqslant \mathrm{nla}^l(C_1)$. Thus, since no recursive call to the bisection procedure is made if $\mathrm{nla}^l(\hat{C}) \in \{\mathrm{nla}^l(C_1), \mathrm{nla}^l(C_2)\}$, at most three cuts with exactly q large arcs can be computed within Algorithm 8 for each $0 \leqslant q \leqslant m$, i.e., there are at most $3(m+1)$ calls of the bisection procedure, which yields a total running time of $\mathcal{O}(m \cdot \mathcal{T}_{\mathrm{MC}})$. □

4.3.1 Approximating NFI on $\mathbf{G}(\mathbf{l}^\star, \bar{\mathbf{u}})$

In order to apply Lemma 4.15 to Algorithm 8, it remains to show that our algorithm meets the lemma's second prerequisite, i.e., that Algorithm 8 applied for $l = l^\star$ returns an α-approximate cut interdiction strategy for NFI on $G(l^\star, \bar{u})$. In this section, we, therefore, consider the case that $l = l^\star$ and introduce the following shorthand notation:

Notation 4.21 For $\gamma \geqslant 1$, we set $G(\gamma) := G(l^\star, \gamma)$ and define $C(\gamma) := C(l^\star, \gamma)$ as the minimum cut in $G(\gamma)$ returned by the deterministic minimum cut algorithm. For a cut C, we also write $\text{nla}(C) := \text{nla}^{l^\star}(C)$ and analogously $\text{csa}(C) := \text{csa}^{l^\star}(C)$. Further, we introduce a shorthand notation for $u_{l^\star,\bar{u}} : R \to \mathbb{Q}$ and denote it by $u^\star$.

Recall that $C^\star$ is a fixed optimal cut that is also minimum after interdiction. The interdiction strategy when attacking $C^\star$ is to remove exactly the B large arcs from the cut. Additionally using Lemma 4.8 with $C^\star$ being minimum after interdiction in $G(\bar{u})$ yields the following observation:

Observation 4.22 For the cut $C^\star$, it holds that:

- $\text{nla}(C^\star) = B$
- $\text{val}^{G(\bar{u})}(C^\star) = \text{icap}^{G(\bar{u})}(C^\star) = \text{csa}(C^\star)$

First of all, we rule out two cases in which it is easy to find an optimal solution for NFI on $G(\bar{u})$.

Lemma 4.23 If $\text{nla}(C(\bar{u})) \geqslant B$, then $C(\bar{u})$ is optimal for NFI on $G(\bar{u})$ and, hence, for NFI on G.

Proof. As $\text{nla}(C(\bar{u})) \geqslant B$, only large arcs are removed when attacking $C(\bar{u})$, which means $u^\star(D_{C(\bar{u})}) = B \cdot \bar{u}$. Since only large arcs are removed when attacking $C^\star$ as well, it also holds that $u^\star(D_{C^\star}) = B \cdot \bar{u}$. This yields

$$\begin{aligned}
\text{val}^{G(\bar{u})}(C(\bar{u})) &\overset{(1)}{\leqslant} \text{icap}^{G(\bar{u})}(C(\bar{u})) \\
&= \text{cap}^{G(\bar{u})}(C(\bar{u})) - u^\star(D_{C(\bar{u})}) \\
&= \text{cap}^{G(\bar{u})}(C(\bar{u})) - B \cdot \bar{u} \\
&\leqslant \text{cap}^{G(\bar{u})}(C^\star) - B \cdot \bar{u} \\
&= \text{cap}^{G(\bar{u})}(C^\star) - u^\star(D_{C^\star}) \\
&= \text{icap}^{G(\bar{u})}(C^\star) \\
&\overset{(2)}{=} \text{val}^{G(\bar{u})}(C^\star),
\end{aligned}$$

where (1) follows from Lemma 4.8 and (2) follows from Observation 4.22. The optimality of $C(\bar{u})$ for NFI on $G(\bar{u})$ follows from the optimality of $C^\star$ and the optimality of $C(\bar{u})$ for NFI on G then follows immediately from Lemma 4.14. □

Lemma 4.24 If $\text{nla}(C(1)) \leqslant B$, then $C(1)$ is optimal for NFI on $G(\bar{u})$ and, hence, for NFI on G.

Proof. The interdiction strategy for attacking the cut $C(1)$ is to first remove all large arcs from the cut, and then spend the remaining budget on removing $B - \text{nla}(C(1))$ small arcs, which all have at least a capacity of one. The assumption that $\text{nla}(C(1)) \leqslant B$ ensures that the budget is large enough to remove all large arcs from the cut. Therefore, it holds that

$$\begin{aligned}
\text{val}^{G(\bar{u})}(C(1)) &\overset{(1)}{\leqslant} \text{icap}^{G(\bar{u})}(C(1)) \\
&\leqslant \text{csa}(C(1)) - (B - \text{nla}(C(1))) \cdot 1 \\
&= \text{csa}(C(1)) + \text{nla}(C(1)) - B \\
&= \text{cap}^{G(1)}(C(1)) - B \\
&\leqslant \text{cap}^{G(1)}(C^\star) - B \\
&= \text{csa}(C^\star) + \underbrace{\text{nla}(C^\star) - B}_{=0 \text{ due to Obs. 4.22}} \\
&= \text{csa}(C^\star) \\
&\overset{(2)}{=} \text{val}^{G(\bar{u})}(C^\star),
\end{aligned}$$

where (1) follows from Lemma 4.8 and (2) follows from Observation 4.22. Since $C^\star$ is optimal for NFI on $G(\bar{u})$ by Corollary 4.13, this shows that $C(1)$ is optimal for NFI on $G(\bar{u})$. The optimality of $C(1)$ for NFI on G then follows from Lemma 4.14. □

Lemmas 4.23 and 4.24 imply that, if Algorithm 8 enters one of the if-statements in line 3 or line 5 for $l = l^\star$, it returns an optimal solution. Hence, we assume in the following that Algorithm 8 enters the else-statement in line 7.

The next lemma states a sufficient condition under which the bisection procedure in Algorithm 9 returns an optimal solution:

Lemma 4.25 If Algorithm 9 finds a value γ such that $\text{nla}(C(\gamma)) = B$, then $C(\gamma)$ is optimal for NFI on $G(\bar{u})$ and, hence, for NFI on G.

Proof. As $\text{nla}(C(\gamma)) = B$, the interdiction strategy for attacking the cut $C(\gamma)$ consists of removing exactly the large arcs from the cut. Therefore, the interdicted capacity of the cut $C(\gamma)$ in the network $G(\bar{u})$ is equal to $\text{csa}(C(\gamma))$. Analogously to the proof of Lemma 4.24, it holds that

$$\begin{aligned}
\text{val}^{G(\bar{u})}(C(\gamma)) &\leqslant \text{icap}^{G(\bar{u})}(C(\gamma)) \\
&= \text{csa}(C(\gamma)) + \underbrace{(\text{nla}(C(\gamma)) - B)}_{=0} \cdot \gamma \\
&= \text{cap}^{G(\gamma)}(C(\gamma)) - B \cdot \gamma \\
&\leqslant \text{cap}^{G(\gamma)}(C^\star) - B \cdot \gamma \\
&= \text{csa}(C^\star) + \gamma \cdot \underbrace{(\text{nla}(C^\star) - B)}_{=0 \text{ due to Obs. 4.22}} \\
&= \text{csa}(C^\star) \\
&= \text{val}^{G(\bar{u})}(C^\star),
\end{aligned}$$

where the first inequality follows from Lemma 4.8 and the last equality follows from Observation 4.22. This proves optimality of $C(\gamma)$ for NFI on $G(\bar{u})$. The optimality for NFI on G follows immediately from Lemma 4.14. $\square$

Due to Lemma 4.25, only the case that the algorithm does not find any cut $C(\gamma)$ with $\text{nla}(C(\gamma)) = B$ has to be considered. In the following, we fix $\mathring{C}_1$ to be the cut in the set bisection$(l^\star, 1, \bar{u})$ with minimum value $\text{nla}(\mathring{C}_1)$ such that $\text{nla}(\mathring{C}_1) > B$, and $\mathring{C}_2$ to be the cut in bisection$(l^\star, 1, \bar{u})$ with maximum value $\text{nla}(\mathring{C}_2)$ such that $B > \text{nla}(\mathring{C}_2)$. Both of these cuts exist as $\text{nla}\,(C(1)) > B$ and $\text{nla}(C(\bar{u})) < B$ since Algorithm 8 entered neither of the if-statements in line 3 or line 5. Note also that both cuts are uniquely defined as no two cuts in bisection$(l^\star, 1, \bar{u})$ can have the same number of large arcs due to Lemma 4.17 and the if-statement in line 5 of Algorithm 9.

Furthermore, we fix $\mathring{\gamma}_1$ and $\mathring{\gamma}_2$ to be the values that led the algorithm to compute $\mathring{C}_1$ and $\mathring{C}_2$, respectively. In particular, this implies that bisection$(l^\star, \mathring{\gamma}_1, \mathring{\gamma}_2)$ has been called during the execution of the algorithm and no further recursive calls have been made during this recursion step. Finally, we fix $\mathring{\gamma}$ to be the value of $\hat{\gamma}$ in the call to bisection$(l^\star, \mathring{\gamma}_1, \mathring{\gamma}_2)$, i.e., $\mathring{\gamma} := {}^{\text{csa}(\mathring{C}_2) - \text{csa}(\mathring{C}_1)}\!/\!_{\text{nla}(\mathring{C}_1) - \text{nla}(\mathring{C}_2)}$.

Lemma 4.26 It holds that $\text{cap}^{G(\mathring{\gamma})}(\mathring{C}_1) = \text{cap}^{G(\mathring{\gamma})}(\mathring{C}_2) \leqslant \text{cap}^{G(\mathring{\gamma})}(C^\star)$.

Proof. Note that $\text{cap}^{G(\mathring{\gamma})}(\mathring{C}_1) = \text{cap}^{G(\mathring{\gamma})}(\mathring{C}_2)$ holds by the construction of $\mathring{\gamma}$. For the sake of a contradiction, suppose that $\text{cap}^{G(\mathring{\gamma})}(\mathring{C}_1) > \text{cap}^{G(\mathring{\gamma})}(C^\star)$. This means that $\mathring{C}_1$ and $\mathring{C}_2$ are not minimum cuts in $G(\mathring{\gamma})$, which implies that the cut $\hat{C} := C(\mathring{\gamma})$ must also fulfill $\text{cap}^{G(\mathring{\gamma})}(\mathring{C}_1) > \text{cap}^{G(\mathring{\gamma})}(\hat{C})$. Due to the choice of $\mathring{C}_1$ and $\mathring{C}_2$, it must hold that $\text{nla}(\hat{C}) \in \{\text{nla}(\mathring{C}_1), \text{nla}(\mathring{C}_2)\}$ because, otherwise, a further recursive call to the bisection procedure would have been made during the execution of bisection $(l^\star, \mathring{\gamma}_1, \mathring{\gamma}_2)$.

If $\text{nla}(\hat{C}) = \text{nla}(\mathring{C}_1)$, it follows from $\text{cap}^{G(\mathring{\gamma})}(\mathring{C}_1) > \text{cap}^{G(\mathring{\gamma})}(\hat{C})$ that $\text{csa}(\mathring{C}_1) > \text{csa}(\hat{C})$, which is a contradiction to $\mathring{C}_1$ being a minimum cut in $G(\mathring{\gamma}_1)$. Analogously, if $\text{nla}(\hat{C}) = \text{nla}(\mathring{C}_2)$, it follows from $\text{cap}^{G(\mathring{\gamma})}(\mathring{C}_2) > \text{cap}^{G(\mathring{\gamma})}(\hat{C})$ that $\text{csa}(\mathring{C}_2) > \text{csa}(\hat{C})$, which is a contradiction to $\mathring{C}_2$ being a minimum cut in $G(\mathring{\gamma}_2)$. $\square$

Proposition 4.27 If Algorithm 8 is applied for $l = l^\star$ and if the algorithm enters the else-statement in line 7, it returns a $(B+1)$-approximate cut interdiction strategy for NFI on $G(\bar{u})$ and, hence, returns a $(B+1)$-approximate cut interdiction strategy for NFI on G.

Proof. By Lemma 4.26, it holds that:

$$\text{cap}^{G(\mathring{\gamma})}(\mathring{C}_1) \leqslant \text{cap}^{G(\mathring{\gamma})}(C^\star) \tag{4.3}$$

$$\text{cap}^{G(\mathring{\gamma})}(\mathring{C}_2) \leqslant \text{cap}^{G(\mathring{\gamma})}(C^\star) \tag{4.4}$$

From (4.3), we obtain

$$\begin{aligned} 0 &\leqslant \text{cap}^{G(\mathring{\gamma})}(C^\star) - \text{cap}^{G(\mathring{\gamma})}(\mathring{C}_1) \\ &= \text{nla}(C^\star) \cdot \mathring{\gamma} + \text{csa}(C^\star) - \text{nla}(\mathring{C}_1) \cdot \mathring{\gamma} - \text{csa}(\mathring{C}_1) \end{aligned}$$

$$\begin{aligned}
&\leqslant \mathrm{nla}(C^\star)\cdot\mathring{\gamma} + \mathrm{csa}(C^\star) - \mathrm{nla}(\mathring{C}_1)\cdot\mathring{\gamma}\\
&= \mathrm{csa}(C^\star) + (\underbrace{\mathrm{nla}(C^\star)}_{=B} - \underbrace{\mathrm{nla}(\mathring{C}_1)}_{\geqslant B+1})\cdot\mathring{\gamma}\\
&\leqslant \mathrm{csa}(C^\star) - \mathring{\gamma}.
\end{aligned}$$

Due to Observation 4.22, it holds that $\mathrm{val}^{G(\bar{u})}(C^\star) = \mathrm{icap}^{G(\bar{u})}(C^\star) = \mathrm{csa}(C^\star)$, which, together with the above inequality, yields

$$\mathrm{val}^{G(\bar{u})}(C^\star) = \mathrm{csa}(C^\star) \geqslant \mathring{\gamma}. \tag{4.5}$$

Similarly, from (4.4), we obtain

$$\begin{aligned}
0 &\geqslant \mathrm{cap}^{G(\mathring{\gamma})}(\mathring{C}_2) - \mathrm{cap}^{G(\mathring{\gamma})}(C^\star)\\
&= (\mathrm{nla}(\mathring{C}_2) - \mathrm{nla}(C^\star))\cdot\mathring{\gamma} + \mathrm{csa}(\mathring{C}_2) - \mathrm{csa}(C^\star)\\
&\overset{(1)}{=} (\mathrm{nla}(\mathring{C}_2) - B)\cdot\mathring{\gamma} + \mathrm{csa}(\mathring{C}_2) - \mathrm{csa}(C^\star)\\
&\overset{(2)}{=} (\mathrm{nla}(\mathring{C}_2) - B)\cdot\mathring{\gamma} + \mathrm{csa}(\mathring{C}_2) - \mathrm{val}^{G(\bar{u})}(C^\star)\\
&\geqslant -B\cdot\mathring{\gamma} + \mathrm{csa}(\mathring{C}_2) - \mathrm{val}^{G(\bar{u})}(C^\star)\\
&\overset{(3)}{\geqslant} -B\cdot\mathring{\gamma} + \mathrm{val}^{G(\bar{u})}(\mathring{C}_2) - \mathrm{val}^{G(\bar{u})}(C^\star)
\end{aligned}\tag{4.6}$$

where (1) and (2) hold due to Observation 4.22. Further, Inequality (3) holds since there are less than B large arcs in the cut $\mathring{C}_2$, meaning that the interdicted capacity of $\mathring{C}_2$ can be at most $\mathrm{csa}(\mathring{C}_2)$. Thus, Lemma 4.8 implies that $\mathrm{val}^{G(\bar{u})}(\mathring{C}_2) \leqslant \mathrm{csa}(\mathring{C}_2)$.
From (4.6), we get that $\mathrm{val}^{G(\bar{u})}(C^\star) + B\cdot\mathring{\gamma} \geqslant \mathrm{val}^{G(\bar{u})}(\mathring{C}_2)$. This together with (4.5) yields

$$(B+1)\cdot\mathrm{val}^{G(\bar{u})}(C^\star) \geqslant \mathrm{val}^{G(\bar{u})}(C^\star) + B\cdot\mathring{\gamma} \geqslant \mathrm{val}^{G(\bar{u})}(\mathring{C}_2).$$

Hence, as $C^\star$ is an optimal solution for NFI on $G(\bar{u})$, the cut $\mathring{C}_2$ is a $(B+1)$-approximate cut for NFI on $G(\bar{u})$. Due to Lemma 4.14, the cut $\mathring{C}_2$ is also a $(B+1)$-approximate cut for NFI on G, which completes the proof. □

We now show the main result of this section:

Proposition 4.28 If Algorithm 8 is applied for $l = l^\star$, it returns a $(B+1)$-approximate solution for NFI on $G(\bar{u})$ and, hence, for NFI on G.

Proof. This follows immediately from Lemmas 4.23 and 4.24 and Proposition 4.27. □

Theorem 4.29 There exists a $(B+1)$-approximation algorithm for NFI on G that runs in $\mathcal{O}(m^2\cdot\mathcal{T}_{\mathrm{MC}})$ time, where $\mathcal{T}_{\mathrm{MC}}$ is the maximum time needed to compute a minimum cut in a network of the form $G(l,\gamma)$ with $l \in \{1,\dots,m-B+1\}$ and $\gamma \in [1,\bar{u}]$.

Proof. Follows from Lemma 4.15, Proposition 4.20, and Proposition 4.28. □

Note that the networks $G(l,\gamma)$ and G only differ in the arc capacities. Hence, if a *strongly polynomial* minimum cut algorithm is used, then $\mathcal{T}_{\text{MC}}$ equals the running time of this algorithm on G. To the best of our knowledge, the currently fastest deterministic algorithm for computing a minimum cut is due to Orlin [Orl13] and runs in $\mathcal{O}(nm)$ time. Thus, using this algorithm, Theorem 4.29 yields a running time of $\mathcal{O}\left(nm^3\right)$ for our algorithm.

Finally, we provide an example where the approximation ratio is almost tight in the sense that the algorithm does not yield a B-approximate solution. The network in this example is a pearl graph (in particular, a series-parallel graph) with only two different arc capacities, which means that our analysis is almost tight even for this special case.

Example 4.30 Let $G = (V,R,u)$ be given by $V = \{s, v_1, v_2, t\}$ and $R = R_1 \cup R_2 \cup R_3$. The set R_1 consists of $(B+1)^2$ parallel arcs from s to v_1 with capacity 1, R_2 consists of B parallel arcs from v_1 to v_2 with capacity 1, and $B+1$ parallel arcs from v_1 to v_2 with capacity $\bar{u} > (B+1)^2$, and R_3 consists of $B+1$ parallel arcs from v_2 to t with capacity $\bar{u}$. This is illustrated in Figure 4.2.

The numbering of the arcs is such that the indices of the arcs in R_1 are lower than the indices of those in R_2, and the indices of the arcs in R_2 are lower than the indices of those in R_3.

This network contains only three cuts. Since $\bar{u} > (B+1)^2$, the cut $C(\bar{u})$ is the cut $(\{s\}, \{v_1, v_2, t\})$, while the unique optimal cut is $C_{\text{OPT}} = (\{s, v_1\}, \{v_2, t\})$. The cut $C(1)$ is the cut $(\{s, v_1, v_2\}, \{t\})$. Note that this yields $l^\star = m - 2B$.

We start by showing that C_{OPT} is not guaranteed to be found by Algorithm 8 when calling `bisection` $(l^\star, 1, \bar{u})$ for $l = l^\star = m - 2B$. Note that we obtain $\hat{\gamma} = B+1$ in the call to `bisection` $(l^\star, 1, \bar{u})$, and all three cuts have the same capacity of $(B+1)^2$ in $G(\hat{\gamma})$. Thus, the deterministic minimum cut algorithm might return any of the three possible cuts, in which case C_{OPT} is not necessarily found by the algorithm – which we assume in the following.

To show that the cut C_{OPT} is indeed not found by the algorithm, it also needs to be shown that it is not found when calling Algorithm 8 for any other value $l \neq l^\star$. To this end, first consider the case where $l < l^\star$. Due to the numbering, this means that C_{OPT} contains at least $B+1$ large arcs. This implies that, for any $\gamma \in [1, \bar{u}]$, it holds that

$$\text{cap}^{G(l,\gamma)}(C_{\text{OPT}}) > \text{cap}^{G(l,\gamma)}((\{s, v_1, v_2\}, \{t\})),$$

which means that the optimal cut is neither found when entering one of the if-statements in line 3 or line 5 of Algorithm 8, nor can it be found by a call to `bisection`$(l, 1, \bar{u})$.

Now consider the case where $l > l^\star$. Then the cut C_{OPT} contains at least one arc of capacity $\bar{u}$ in $G(l,\gamma)$ for any $\gamma \in [1, \bar{u}]$. Since $\bar{u} > (B+1)^2$, this means that

$$\text{cap}^{G(l,\gamma)}(C_{\text{OPT}}) > \text{cap}^{G(l,\gamma)}((\{s\}, \{v_1, v_2, t\})),$$

which, again, implies that the optimal cut is neither found when entering one of the if-statements in line 3 or line 5 of Algorithm 8, nor can it be found by a call to `bisection`$(l, 1, \bar{u})$.

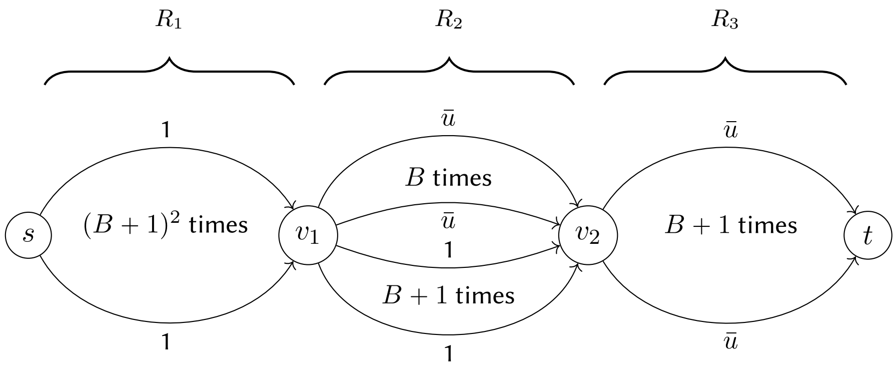


Figure 4.2.: Network of the instance described in Example 4.30.

For the values of the cuts, we have

$$\begin{aligned}\text{val}^G((\{s,v_1,v_2\},\{t\})) &= \min\{\bar{u},(B+1)^2\} = \bar{u},\\ \text{val}^G(C_{\text{OPT}}) &= B+1, \text{ and}\\ \text{val}^G((\{s\},\{v_1,v_2,t\})) &= B^2+B+1.\end{aligned}$$

Thus, since $\bar{u} > (B+1)^2$, the interdiction strategy obtained by attacking the cut $(\{s\},\{v_1,v_2,t\})$ is returned by the algorithm and $\text{val}^G((\{s\},\{v_1,v_2,t\})) = B^2+B+1 > B^2+B = B\cdot\text{val}^G(C_{\text{OPT}})$, which shows that the algorithm is not a B-approximation algorithm for NFI on G.

4.4 Further Reducing the Approximation Ratio

In this section, we present an algorithmic approach how to further reduce the approximation ratio of an approximation algorithm for NFI whose approximation ratio depends on the budget B. The idea is to fix a number $k \leqslant B$ and guess a set $\hat{D} \subseteq R$ consisting of k arcs that are removed in an optimal solution $D_{\text{OPT}} \subseteq R$. The presented algorithm tries this for all $\binom{m}{k}$ possible combinations of k arcs, which means it will actually guess correctly at least once. For each guess, it removes the arcs from the network G and applies the given approximation algorithm to the network $G_{\hat{D}}$ with the remaining budget of $B-k$. The solution returned by the approximation algorithm together with the arcs in $\hat{D}$ yields a set of exactly B arcs, i.e., a feasible interdiction strategy for the original problem. Similar to the idea of Lemma 4.15, the algorithm chooses the solution with lowest value out of all $\binom{m}{k}$ computed solutions.

Let G be the input network and B the budget for an instance of NFI. In the following, we denote by $\text{ALG}(G,B) \subseteq R$ the output of a polynomial-time approximation algorithm for NFI on the given instance with approximation ratio $\alpha(B)$, where we assume that $\alpha(B)$ is increasing in B. The algorithm described above is stated in Algorithm 10.

The following proposition shows how the approximation ratio can be improved by applying Algorithm 10:

Algorithm 10: BOOST-BY-GUESS

```
1 Procedure boost-by-guess($G, B, k$)
2     $\mathcal{R} = \varnothing$
3     forall $\hat{D} \subseteq R$ with $|\hat{D}| = k$ do
4         $\mathcal{R} = \mathcal{R} \cup \{\hat{D} \cup \text{ALG}(G_{\hat{D}}, B - k)\}$
5     return $\operatorname{argmin}_{D \in \mathcal{R}} \text{val}^G(D)$
```

Proposition 4.31 For constant $k \leqslant B$, Algorithm 10 returns an $\alpha(B-k)$-approximate solution for NFI on G with budget B.

Proof. Let D_{OPT} be an optimal solution and let $\hat{D} \subseteq D_{\text{OPT}}$ with $|\hat{D}| = k$. Furthermore, let $\hat{G} := G_{\hat{D}}$. Then, for any set $\bar{D}$ of $B-k$ arcs in $\hat{G}$, it holds that $G_{\hat{D} \cup \bar{D}} = \hat{G}_{\bar{D}}$. Hence, a minimum cut in $G_{\hat{D} \cup \bar{D}}$ is also a minimum cut in $\hat{G}_{\bar{D}}$. Due to Observation 4.2, the value of the solution $\bar{D}$ on $\hat{G}$ using budget $B-k$ is the same as the value of the solution $\hat{D} \cup \bar{D}$ on G using budget B. In particular, choosing $\bar{D} = D_{\text{OPT}} \backslash \hat{D}$ implies that $D_{\text{OPT}} \backslash \hat{D}$ must be an optimal solution for NFI on $\hat{G}$ with budget $B-k$. This yields

$$\begin{aligned}
\text{val}^G(\hat{D} \cup \text{ALG}(\hat{G}, B-k)) &= \text{val}^{\hat{G}}(\text{ALG}(\hat{G}, B-k)) \\
&\leqslant \alpha(B-k) \cdot \text{val}^{\hat{G}}(D_{\text{OPT}} \backslash \hat{D}) \\
&= \alpha(B-k) \cdot \text{val}^G(D_{\text{OPT}}).
\end{aligned}$$

Now, since $\hat{D} \cup \text{ALG}(\hat{G}, B-k) = \hat{D} \cup \text{ALG}(G_{\hat{D}}, B-k)$ is added to the set $\mathcal{R}$ during the for loop starting in line 3 of Algorithm 10 and the algorithm returns a solution of minimum value among all solutions in $\mathcal{R}$, the solution returned by Algorithm 10 is as least as good as $\hat{D} \cup \text{ALG}(\hat{G}, B-k)$ and, therefore, is an $\alpha(B-k)$-approximate solution for NFI on G with budget B. □

Plugging Algorithm 8 from the previous chapter into the proposition directly yields the following theorem:

Theorem 4.32 There exists a polynomial-time $(B-k+1)$-approximation algorithm for NFI for any constant $k \leqslant B$.

In particular, this implies that, if B is constant, then there exists a polynomial-time exact algorithm for NFI on G with budget B. This should, however, not be too surprising since using $k = B$ in Algorithm 10 translates to enumerating all possible solutions (which are polynomially many if B is constant).

4.5 Conclusion

In this chapter, a $(B+1)$-approximation algorithm for NFI with unit removal costs is presented. To the best of our knowledge, the proposed approximation algorithm is the first approximation algorithm for any variant of NFI whose approximation ratio only depends on the budget

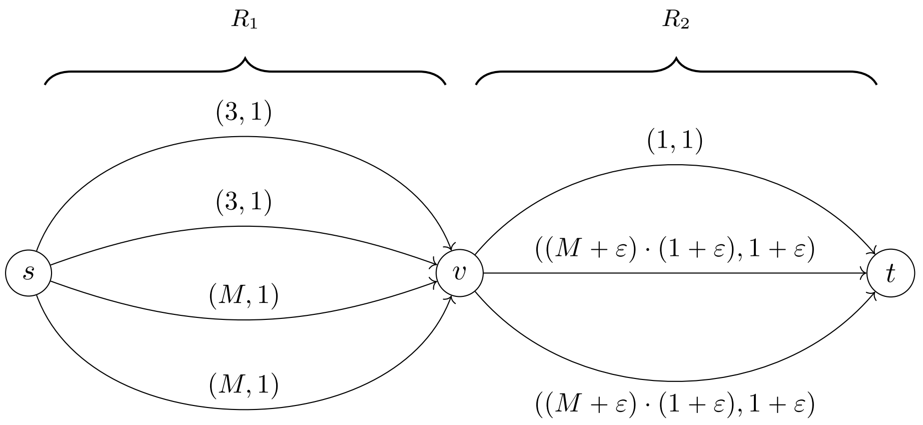

R_1
R_2
$(3,1)$
$(3,1)$
$(M,1)$
$(M,1)$
$(1,1)$
$((M+\varepsilon)\cdot(1+\varepsilon),1+\varepsilon)$
$((M+\varepsilon)\cdot(1+\varepsilon),1+\varepsilon)$
s
v
t

Figure 4.3.: Network of the instance described in Example 4.33 where the capacities and the removal costs are the first and second argument in the brackets, respectively.

available to the interdictor, but not on the size of the network. Especially in the case of simple graphs, where we can assume that $B < n-1$, this is a significant improvement over the previously best known approximation ratio of $n-1$.

It is also worth noting that the minimum cut algorithm used as a subroutine within our algorithm is only required to be deterministic because it should return the same cut when applying it twice to the same network. In an efficient implementation of our algorithm, however, it is anyhow advisable to store the already computed cuts, in which case our results would still hold true with the use of a non-deterministic minimum cut algorithm.

Further, throughout this chapter, we never make use of the network being directed, which is why all our results also hold true for NFI on undirected networks.

An obvious open question is whether our algorithm can be extended to the general version of NFI, where arcs may have different costs. In this case, the best strategy for attacking a given cut does no longer consist of removing the arcs of largest capacity, but its computation requires solving a (weakly) $\mathcal{NP}$-hard (minimization) knapsack problem. A greedy approximation algorithm for the knapsack problem is used in [CZ17] to extend the case of unit removal costs. To this end, the arcs are sorted by their efficiency $\rho(r) := {}^{u(r)}\!/_{c(r)}$ and the algorithm for unit removal costs is applied using the obtained sorting of the arcs. However, this approach does not work for our algorithm as the following example shows.

Example 4.33 Let $B=2$ and let $G=(V,R,u)$ be given by $V=\{s,v,t\}$ and $R=R_1\cup R_2$. Further, let $1 << M$ and let $\varepsilon := {}^1\!/_M$. The set R_1 consists of two parallel arcs from s to v with capacity 3 and removal cost 1 and another two parallel arcs from s to v with capacity M and removal cost 1. The set R_2 consists of an arc from v to t with capacity 1 and removal cost 1 and two parallel arcs from v to t with capacity $(M+\varepsilon)\cdot(1+\varepsilon)$ and removal cost $1+\varepsilon$. The network is illustrated in Figure 4.3. For every value of l, our algorithm only finds the cut $C_2 := (\{s,v\},\{t\})$ but not the cut $C_1 := (\{s\},\{v,t\})$. However, the value for NFI when attacking C_1 is 6 while the value for NFI of attacking C_2 is $(M+\varepsilon)\cdot(1+\varepsilon)+1$. Hence, our algorithm with the technique used in [CZ17] cannot achieve a bounded approximation ratio.

5 Complexity of the Temporal Shortest Path Interdiction Problem

Abstract In the shortest path interdiction problem, an interdictor aims to remove arcs of total cost at most a given budget from a directed graph with given arc costs and traversal times such that the length of a shortest s-t-path is maximized. For static graphs, this problem is known to be strongly $\mathcal{NP}$-hard, and it has received considerable attention in the literature.
However, the shortest path interdiction problem has not yet been formally studied on temporal graphs – a graph class where arcs are only available at certain times. Here, common definitions of a "shortest path" include: *latest start path* (path with maximum start time), *earliest arrival path* (path with minimum arrival time), *shortest duration path* (path with minimum traveling time including waiting times at nodes), and *shortest traversal path* (path with minimum traveling time *not* including waiting times at nodes).
In this chapter, we analyze the complexity of the shortest path interdiction problem on temporal graphs with respect to all four definitions of a shortest path mentioned above. Even though the shortest path interdiction problem on static graphs is known to be strongly $\mathcal{NP}$-hard, we show that the latest start and the earliest arrival path interdiction problems on temporal graphs are polynomial-time solvable. For the shortest duration and the shortest traversal path interdiction problem, however, we show strong $\mathcal{NP}$-hardness, but we obtain polynomial-time algorithms for these problems on extension-parallel temporal graphs.
Finally, we show how the complexities of the problems change under three slightly modified versions of the problem.

5.1 Introduction

Not least because of its great applicability to a wide range of real-world problems, the shortest s-t-path problem is undeniably one of the most central and well-studied problems in graph theory and network optimization. It is, hence, not surprising that the *shortest path interdiction problem*, where arcs are to be removed from a graph subject to a given budget such that the length of a shortest s-t-path for two given nodes s and t is maximized, is one of the most relevant interdiction problems. On static graphs, where the graph is not subject to change over time, this problem is widely studied. The assumption of a graph not changing over time, however, is often too restrictive when modeling real-world problems such as, e.g., the spread of the virus during the COVID-19 pandemic. In such settings, the concept of *temporal graphs*, where arcs are only available at certain times, allows for more realistic models (see, e.g., [BRP21; Enr+21]) and has recently attracted the interest of researchers in algorithmic network optimization (see, e.g., [Akr+20; MS16; Mol20]).

In this chapter, we investigate the *temporal shortest path interdiction problem*, where the aim is to remove arcs from a directed temporal graph such that the length of a shortest path from a node s to another node t is maximized. As the length of a path in a temporal graph can be interpreted in various different ways, we investigate four common versions of the temporal shortest path interdiction problem. We show that two of these versions are polynomial-time solvable, while the other two are strongly $\mathcal{NP}$-hard.

5.1.1 Previous Work

The following paragraphs summarize the state-of-the-art concerning shortest path problems on temporal graphs, the (static) shortest path interdiction problem, and related interdiction problems on temporal graphs.

We start with an overview of the literature about shortest path problems on temporal graphs. The model of a temporal graph used in this chapter (and, e.g., in [BFJ03; Wu+16]) is sometimes also referred to as a *scheduled network* [Ber96] or a *point-availability time-dependent network* [BCV21]. Here, each temporal arc r can only be entered at a given start time $\tau(r)$ and it takes $\lambda(r)$ units of time to traverse the arc, which leads to an arrival time of $\tau(r) + \lambda(r)$ at the end node of the arc. In this model, four different definitions of a "shortest path" between two nodes s and t are considered (see [Wu+16]):

- *reverse-foremost* or *latest start* path, which is an s-t-path with maximum start time of the first arc in the path,
- *foremost* or *earliest arrival* path, which is an s-t-path with minimum arrival time of the last arc in the path,
- *shortest duration* path, which is an s-t-path with minimum total traveling time including waiting times at the nodes,
- *shortest traversal* path, which is an s-t-path with minimum total traveling time *not* including waiting times at the nodes.

For each of the four definitions, the corresponding temporal shortest path problem can be solved efficiently, i.e., a shortest path can be computed in polynomial time [Ben+20; BFJ03; Wu+16].

A different definition of temporal graphs is considered, e.g, in [Mol20], where a wide range of well-studied graph problems is investigated on temporal graphs. This definition can be interpreted as the special case of the previous definition obtained when all traversal times are zero.[1] Bi-objective versions of temporal shortest path problems are considered in [BCV21; MO19; Oet22].

A definition that allows for continuous availability of arcs in a temporal graph as well as a time dependency of an arc's traversal time is provided in [Cas+12]. This definition can be seen as a generalization of the definition from [BFJ03; Wu+16] used here. However, due to the definition's large generality, it does not allow for a finite encoding of temporal graphs without imposing further assumptions, so classical techniques of complexity analysis cannot be applied for the most general form of this definition. A natural finite encoding is possible, e.g., if each arc is restricted to be present over a time interval, i.e., it can be entered at any time between two specified points in time. Even for this special case of the definition in [Cas+12], it is shown in Section 5.4.2 that deciding whether two nodes s and t can be separated by removing no more than B arcs from the graph is already strongly $\mathcal{NP}$-hard.

Next, the literature about the shortest path interdiction problem on static graphs is summarized. To explicitly distinguish between the problem on *static* graphs and the problem on *temporal* graphs, we refer to the shortest path interdiction problem on static graphs as the *static* shortest path interdiction problem (S-SP-IP) in the following. This problem is also referred to as the *most vital arcs problem* in the literature [BGV89]. S-SP-IP is one of the most-studied network interdiction problems and a vast amount of literature exists on the problem. A detailed overview is provided in [SS20]. Concerning the complexity of S-SP-IP, the first proof of weak $\mathcal{NP}$-hardness is provided in [BGV89]. This result is extended in [BKS95], where it is shown that S-SP-IP is strongly $\mathcal{NP}$-hard even on acyclic graphs and for the special case of unit arc lengths and removal costs. This result is further extended in [Kha+08], where it is shown that it is $\mathcal{NP}$-hard to approximate S-SP-IP within any factor $\alpha < 2$. Indeed, it is still an open question whether any non-trivial approximation algorithms exist for S-SP-IP. Variations of S-SP-IP considering online settings, randomized interdiction strategies, or multiple objectives have recently been studied, e.g., in [BSR20; HS21; SS16].

While, to the best of our knowledge, the complexity of the shortest path interdiction problem has not been formally investigated on temporal graphs, a polynomial-time algorithm that decides whether there exist k arc-disjoint temporal s-t-paths is presented in [Ber96]. They further show that it can be decided in polynomial time whether there exists a temporal s-t-path arriving before a given arrival time even if up to k arcs are removed, which implicitly solves the temporal earliest arrival path interdiction problem for unit removal costs. However, it is not clear whether the algorithm can be extended to the case in which arcs can have different removal costs.

Further, related reachability interdiction problems on temporal graphs are studied in [DP22; EMS21; Enr+21; MRZ21]. Here, the goal is to minimize (or maximize in some cases) the number of nodes reachable from a single node or a set of nodes in a temporal graph by either removing arcs, delaying start times, or changing the order of start times. While the vast majority of studied problems turn out to be $\mathcal{NP}$-hard even under severe restrictions, only a few special cases are shown to be polynomial-time solvable. Moreover, the problem of separating two given

[1]This implies that all polynomial-time solvability results presented here can immediately be transferred to the definition used in [Mol20]. For our hardness results, we point out explicitly whether they can be transferred to this definition of temporal graphs.

nodes by removing nodes from a temporal graph is considered for various settings in [Flu+20; Ibi+22; KKK00; Maa+23; Mol22; Zsc+20]. Again, most of the problems are $\mathcal{NP}$-hard, while some polynomial-time solvability results – mostly for specific classes of graphs – are shown.

5.1.2 Our Contribution

We analyze the complexity of the shortest path interdiction problem on temporal graphs with respect to all four definitions of a shortest path considered in [Wu+16]. Even though S-SP-IP is known to be strongly $\mathcal{NP}$-hard, it is found that polynomial-time algorithms for the latest start and the earliest arrival path interdiction problem on temporal graphs exist. These algorithms exploit the fact that, for these versions of the problem, the objective value of a path only depends on either the first or on the last arc in the path (but not on both). For the shortest duration and shortest traversal path interdiction problem, where both the first and the last arc in a path (and the amount of time spend waiting at nodes in the former case) are relevant for its length, however, we show strong $\mathcal{NP}$-hardness. Our reduction further implies that, unless $\mathcal{P} = \mathcal{NP}$, there exists no polynomial-time approximation algorithm for any of the two problems with an approximation ratio smaller than $3/2$.

On extension-parallel temporal graphs, however, we obtain polynomial-time algorithms for the shortest duration path interdiction problem and the shortest traversal path interdiction problem. This result can be transferred to the static shortest path interdiction problem, where it also represents a new result.

5.2 Problem Definition

A directed (discrete-time) *temporal graph* G consists of a nonempty, finite set V of nodes and a finite set R of *temporal arcs*. As usual, we denote the number of nodes and the number of (temporal) arcs in the graph by n and m, respectively. A temporal arc $r \in R$ has four attributes, namely its *start node* $\alpha(r) \in V$, its *end node* $\omega(r) \in V$, its *start time* $\tau(r) \in \mathbb{Q}$, and its *traversal time* $\lambda(r) \in \mathbb{Q}_{\geqslant 0}$. When traversing a temporal arc $r \in R$, the *arrival time* of r is $\tau(r) + \lambda(r)$. A *temporal path* $P = (r_1, \ldots, r_k)$ is a sequence of temporal arcs such that, for each $i \in \{1, \ldots, k-1\}$, it holds that $\omega(r_i) = \alpha(r_{i+1})$ and $\tau(r_i) + \lambda(r_i) \leqslant \tau(r_{i+1})$, i.e., the end node of each arc is the start node of the next arc in the path and the arrival time of each arc is less than or equal to the start time of the next arc.[2] For two nodes $s, t \in V$, a temporal path $P = (r_1, \ldots, r_k)$ is called a *(temporal) s-t-path* if $\alpha(r_1) = s$ and $\omega(r_k) = t$. Given a temporal graph $G = (V, R)$, the *underlying static graph* $G^{\text{stat}} = (V^{\text{stat}}, R^{\text{stat}})$ is the (directed) static graph with the same nodes and arcs obtained by disregarding the start times and traversal times of the arcs. A temporal graph is called *acyclic* if its underlying static graph is acyclic, i.e., its underlying static graph does not contain any directed cycle.

While the notion of a "shortest" s-t-path is straightforward in static graphs, temporal graphs allow for various interpretations of the term "shortest". In this chapter, we study the four quality measures for s-t-paths that are presented in [Wu+16].

[2]Note that this definition allows a path to visit the same node (or even traverse the same arc) several times. Except for some results obtained for extensions of the problem in Section 5.4, however, all our results also hold when restricting to *elementary* paths that do not visit any node more than once.

Definition 5.1 Let G be a temporal graph, $s \neq t$ two nodes in G, and $P = (r_1, \ldots, r_k)$ a temporal s-t-path.

- The *start time* of P is defined as $\text{start}(P) := \tau(r_1)$.
- The *arrival time* of P is defined as $\text{arriv}(P) := \tau(r_k) + \lambda(r_k)$.
- The *duration* of P is defined as $\text{dura}(P) := \text{arriv}(P) - \text{start}(P)$.
- The *traversal time* of P is defined as $\text{trav}(P) := \sum_{i=1}^{k} \lambda(r_i)$.

Definition 5.2 Let G be a temporal graph and $s \neq t$ two nodes in G.

- A *latest start path* is an s-t-path with maximum start time. The *latest start time* in G, denoted by $\text{LS}(G)$, is defined as the start time of a latest start path in G.
- An *earliest arrival path* is an s-t-path with minimum arrival time. The *earliest arrival time* in G, denoted by $\text{EA}(G)$, is defined as the arrival time of an earliest arrival path in G.
- A *shortest duration path* is an s-t-path with minimum duration. The *shortest duration* in G, denoted by $\text{SD}(G)$, is defined as the duration of a shortest duration path in G.
- A *shortest traversal path* is an s-t-path with minimum traversal time. The *shortest traversal time* in G, denoted by $\text{ST}(G)$, is defined as the traversal time of a shortest traversal path in G.

If no s-t-path exists in G, $\text{LS}(G)$ is set to $-\infty$, whereas $\text{EA}(G)$, $\text{SD}(G)$, and $\text{ST}(G)$ are set to $+\infty$.

As a side remark, earliest arrival paths are called foremost paths and latest start paths are called reverse-foremost paths in [Wu+16]. Next, the four versions of the temporal shortest path interdiction problem are defined.

Definition 5.3 For an objective OBJ $\in$ {LS, EA, SD, ST}, the temporal OBJ interdiction problem (T-OBJP-IP) is defined as follows.

T-OBJP-IP

INSTANCE:	A temporal graph $G = (V, R)$, two nodes $s \neq t$ in G, a budget $B \in \mathbb{Q}_{>0}$, and removal costs $c : R \to \mathbb{Q}_{\geqslant 0}$
TASK:	Find a subset $D \subseteq R$ of arcs with $\sum_{r \in D} c(r) \leqslant B$ such that $\text{OBJ}(G_D)$ is maximized (minimized in the case that OBJ = LS), where $G_D := (V, R \backslash D)$.

As in Chapter 4, a solution $D \subseteq R$ of T-OBJP-IP with $\sum_{r \in D} c(r) \leqslant B$ is called an *interdiction strategy* and the arcs in D are called *interdicted.* Further, if no temporal path from a node u to another node v exists after the arcs in D have been removed, we say that the interdiction strategy D *separates* u from v or that the pair (u, v) is *separated* by D.

5.3 Polynomial-Time Algorithms and Complexity Results

In this section, we analyze the complexity of each of the four introduced versions of temporal shortest path interdiction. It is shown that two versions can be solved in polynomial time and the other two versions are strongly $\mathcal{NP}$-hard. On extension-parallel temporal graphs, however, the two hard versions are shown to be solvable in polynomial time.

5.3.1 Temporal Latest Start Interdiction

We start by presenting a polynomial-time algorithm to solve T-LSP-IP. This is a surprising result as the static shortest path interdiction problem is known to be strongly $\mathcal{NP}$-hard [BKS95]. The main reason for the polynomial-time solvability of T-LSP-IP is that the obtained objective value only depends on the first arc that is used by a latest start path in the interdicted graph G_D.

In this section, we let $\tau_1 < \tau_2 < \cdots < \tau_l$ denote the distinct start times of outgoing arcs of s in G sorted in increasing order. Further, for $k \in \{1, \ldots, l\}$, we define $G^{\mathrm{LS},k}$ as the temporal graph that results from G by removing all outgoing arcs of s with start time at most τ_k. For completeness, we also define $G^{\mathrm{LS},0} := G$. Our algorithm is based on the following proposition.

Proposition 5.5 Let $k \in \{1, \ldots, l\}$. There exists an interdiction strategy D^k that separates s from t in $G^{\mathrm{LS},k}$ if and only if there exists an interdiction strategy D in G with objective value at most τ_k.

Proof. Let D^k be an interdiction strategy that separates s from t in $G^{\mathrm{LS},k}$. Then, after interdicting the same set $D := D^k$ of arcs in G, no s-t-path in G_D can start with an arc with start time strictly larger than τ_k (otherwise, the path would also be an s-t-path in $G^{\mathrm{LS},k}_{D^k}$). Hence, all s-t-paths in G_D have start time at most τ_k, i.e., D has objective value at most τ_k.

Conversely, let D be an interdiction strategy in G with objective value at most τ_k. Then, no s-t-path in G_D can have start time strictly larger than τ_k, so the interdiction strategy $D^k := D \cap R^{\mathrm{LS},k}$, where $R^{\mathrm{LS},k}$ is the arc set of $G^{\mathrm{LS},k}$, separates s from t in $G^{\mathrm{LS},k}$. □

The idea of the algorithm is to use binary search in order to find $k^\star \in \{1, \ldots, l\}$ such that s can be separated from t in $G^{\mathrm{LS},k^\star}$, but s cannot be separated from t in $G^{\mathrm{LS},k^\star-1}$. Such a $k^\star$ exists whenever s cannot already be separated from t in the whole graph $G = G^{\mathrm{LS},0}$, i.e., whenever the optimal objective value is not equal to $-\infty$. Consequently, in order to obtain a polynomial-time algorithm for T-LSP-IP, it only remains to show that deciding whether a node s can be separated from another node t with a given interdiction budget in an arbitrary temporal graph is possible in polynomial time.

In a static graph, this question can be answered easily by computing a minimum s-t-cut with respect to the removal costs and comparing its total cost to the given interdiction budget B. Hence, we now describe how the question in an arbitrary temporal graph $H = (V, R)$ can be reduced to the static case. To this end, we use a graph construction that is similar to [Wu+16] and to the construction of time-expanded networks in the context of dynamic flows [Orl84]. The constructed graph is therefore called the *time-expanded graph* of H and denoted by $H^{\mathrm{te}} = (V^{\mathrm{te}}, R^{\mathrm{te}})$. We start by defining the set of *crucial times* by $T := \cup_{r \in R}\{\tau(r), \tau(r) + \lambda(r)\}$. For easier notation, we write $T = \{\phi_1, \dots, \phi_j\}$, where the crucial times are indexed in increasing order. For each $v \in V$ and $\phi \in T$, there exists a node (v, ϕ) in V^{te}. For each $i \in \{1, \dots, j-1\}$ and for each $v \in V$, there exists an arc from (v, ϕ_i) to (v, ϕ_{i+1}) with removal cost $B + 1$ (i.e., it cannot be interdicted). Traversing this arc represents waiting at node v of the temporal graph until the next crucial time. Further, for each arc $r \in R$, there exists an arc in R^{te} from $(\alpha(r), \tau(r))$ to $(\omega(r), \tau(r) + \lambda(r))$ with removal cost $c(r)$, whose traversal represents traversing arc r in the temporal graph. We define $s^{\text{te}} := (s, \phi_1)$ and $t^{\text{te}} := (t, \phi_j)$. If the temporal graph H has n nodes and m arcs, its time-expanded graph has $n \cdot |T| \in \mathcal{O}(n \cdot m)$ nodes and $n \cdot (|T| - 1) + m \in \mathcal{O}(n \cdot m)$ arcs. Hence, the size of the time-expanded graph is polynomial in the size of the temporal graph (in contrast to time-expanded networks used in the context of dynamic flows). The following observation follows directly from the construction of H^{te}.

Observation 5.6 There exists an interdiction strategy separating s from t in H if and only if there exists an interdiction strategy separating s^{te} from t^{te} in H^{te}.

Applying the previously described algorithm together with Proposition 5.5 and Observation 5.6 yields the main theorem of this section.

Theorem 5.7 There exists a polynomial-time algorithm for T-LSP-IP with running time in $\mathcal{O}(\log(l) \cdot T_{\text{MC}}(n \cdot m, n \cdot m))$, where $l \leqslant m$ is the number of distinct start times of outgoing arcs of s and $T_{\text{MC}}(n \cdot m, n \cdot m)$ is the time required to compute a minimum s-t-cut in a static graph with $n \cdot m$ nodes and $n \cdot m$ arcs.

5.3.2 Temporal Earliest Arrival Interdiction

In this section, we present a polynomial-time algorithm to solve T-EAP-IP. Similar to T-LSP-IP, the reason for the problem's polynomial-time solvability is that the obtained objective value only depends on the last arc that is used by an earliest arrival path in the interdicted graph G_D. Indeed, an instance of T-EAP-IP can be transformed into an equivalent instance of T-LSP-IP by inverting the direction of all arcs and adjusting the start times and traversal times appropriately. This is described in the following.

Let $G = (V, R)$ be the temporal graph in an instance of T-EAP-IP. We construct a graph $G^{\text{LS}} = (V, R^{\text{LS}})$ for an instance of T-LSP-IP. The *maximum arrival time* in G is defined as $\Phi := \max_{r \in R} \tau(r) + \lambda(r)$. For each $r \in R$, an arc r' is added to R^{LS} with $\alpha(r') := \omega(r)$, $\omega(r') := \alpha(r)$, $\tau(r') := \Phi - \tau(r) - \lambda(r)$, and $\lambda(r') := \lambda(r)$. The arcs r and r' are called *associated.* Further, an interdiction strategy D in G and the interdiction strategy D' in G^{LS} consisting of the arcs in G^{LS} that are associated with those in D are also called *associated.* Defining $s^{\text{LS}} := t$ and $t^{\text{LS}} := s$, $B^{\text{LS}} := B$, and $c^{\text{LS}}(r') := c(r)$ for each pair of associated arcs r and r', it is then easy to see that mapping an interdiction strategy D in G to its associated interdiction strategy D' in G^{LS} defines a bijection between the sets of interdiction strategies in the two graphs. In the following, the instance of T-EAP-IP is denoted by (G, s, t) and the constructed instance of T-LSP-IP by $(G^{\text{LS}}, s^{\text{LS}}, t^{\text{LS}})$. We next show that there is a one-to-one correspondence between temporal paths in G and G^{LS}.

Proposition 5.8 Let r_i and r'_i be associated arcs for each $i \in \{1, \dots, k\}$, and let D and D' be associated interdiction strategies in G and G^{LS}, respectively. Then $P' = (r'_1, \dots, r'_k)$ is a temporal s^{LS}-t^{LS}-path in $G^{\mathrm{LS}}_{D'}$ if and only if $P = (r_k, \dots, r_1)$ is a temporal s-t-path in G_D.

Proof. If $P' = (r'_1, \dots, r'_k)$ is a temporal s^{LS}-t^{LS}-path in $G^{\mathrm{LS}}_{D'}$, then $r'_i \notin D'$ for $i = 1, \dots, k$. Hence, since D' and D are associated, we obtain that $r_i \notin D$ for $i = 1, \dots, k$. Moreover, $t = s^{\mathrm{LS}} = \alpha(r'_1) = \omega(r_1)$, $s = t^{\mathrm{LS}} = \omega(r'_k) = \alpha(r_k)$, and for each $i \in \{1, \dots, k-1\}$, we have $\alpha(r_i) = \omega(r'_i) = \alpha(r'_{i+1}) = \omega(r_{i+1})$ and

$$\tau(r_{i+1}) + \lambda(r_{i+1}) = \Phi - \tau(r'_{i+1}) \leqslant \Phi - \tau(r'_i) - \lambda(r'_i) = \Phi - \tau(r'_i) - \lambda(r_i) = \tau(r_i).$$

Thus, $P = (r_k, \dots, r_1)$ is a temporal s-t-path in G_D as claimed. The inverse direction can be shown along the same lines. □

We call paths P and P' as in Proposition 5.8 *associated* in the following. Proposition 5.8 allows us to show the following relationship between the objective values of associated interdiction strategies for (G, s, t) and $(G^{\mathrm{LS}}, s^{\mathrm{LS}}, t^{\mathrm{LS}})$.

Corollary 5.9 An interdiction strategy D for (G, s, t) has objective value z for T-EAP-IP if and only if the associated interdiction strategy D' for $(G^{\mathrm{LS}}, s^{\mathrm{LS}}, t^{\mathrm{LS}})$ has objective value $\Phi - z$ for T-LSP-IP.

Proof. Given an interdiction strategy D with objective value z and its associated interdiction strategy D', let P be an earliest arrival path in G_D. Then, P has arrival time z and by Proposition 5.8, the associated path P' is a temporal path in $G^{\mathrm{LS}}_{D'}$, whose start time is $\Phi - z$. For the sake of a contradiction, suppose that there exists a path $\bar{P}'$ in $G^{\mathrm{LS}}_{D'}$ with start time $\Phi - \bar{z} > \Phi - z$. Then, by Proposition 5.8, the path $\bar{P}$ that is associated to $\bar{P}'$ is a temporal path in G_D and its arrival time is $\bar{z} < z$, which is a contradiction to P being an earliest arrival path in G_D. Hence, the interdiction strategy D' for $(G^{\mathrm{LS}}, s^{\mathrm{LS}}, t^{\mathrm{LS}})$ has objective value $\Phi - z$. The inverse direction can be shown along the same lines. □

Corollary 5.9 immediately yields the following result.

Corollary 5.10 An interdiction strategy D is optimal for (G, s, t) if and only if its associated interdiction strategy D' is optimal for $(G^{\mathrm{LS}}, s^{\mathrm{LS}}, t^{\mathrm{LS}})$.

Corollary 5.10 and the algorithm presented in Section 5.3.1 yield the main result of this section.

Theorem 5.11 There exists a polynomial-time algorithm for T-EAP-IP with running time in $\mathcal{O}(\log(l) \cdot T_{\mathrm{MC}}(n \cdot m, n \cdot m))$, where $l \leqslant m$ is the number of distinct arrival times of incoming arcs of t and $T_{\mathrm{MC}}(n \cdot m, n \cdot m)$ is the time required to compute a minimum s-t-cut in a static graph with $n \cdot m$ nodes and $n \cdot m$ arcs.

5.3.3 Temporal Shortest Duration Interdiction and Temporal Shortest Traversal Interdiction

In this section, we show that T-SDP-IP and T-STP-IP are strongly $\mathcal{NP}$-hard, even for unit removal costs and if the underlying static graph is acyclic. Moreover, the reduction implies an inapproximability result. We also show, however, that both problems are solvable in polynomial time if the graph is extension-parallel. This result is also shown for the static problem S-SP-IP.

The proof of strong $\mathcal{NP}$-hardness is similar to the proof in [Ben11], where it is shown that finding a multicut in directed acyclic graphs is $\mathcal{APX}$-hard. The reduction is performed from the strongly $\mathcal{NP}$-hard MAX2SAT problem, which is defined as follows.

MAX2SAT

Instance:	A set $X = \{x_1, \dots, x_\zeta\}$ of boolean variables, a set $C = \{c_1, \dots, c_\mu\}$ of clauses each containing two literals, and a positive integer $\delta < \mu$
Question:	Is there a truth assignment for the variables that satisfies at least δ clauses?

Given an instance of MAX2SAT, we construct a temporal graph with removal costs and a corresponding budget. This graph has the property that no s-t-path waits in any node except for s and t, which means that, for each feasible interdiction strategy, the objective values in T-SDP-IP and T-STP-IP are identical. Hence, the resulting instances of T-SDP-IP and T-STP-IP are equivalent in this case. Thus, we present the construction and the corresponding proofs only for T-SDP-IP in the following. An example for the construction is provided in Figure 5.1.

Unless explicitly stated otherwise, all arcs within this construction have start time 0, traversal time 0, and removal cost $B + 1$ (i.e., they cannot be interdicted). We show later that only a slight modification of the construction is necessary in the case of unit removal costs. For each variable $x_i \in X$, there is a *variable gadget* consisting of a directed path with trace $(u_{i,1}, u_{i,2}, u_{i,3}, u_{i,4})$ where only the arcs from $u_{i,1}$ to $u_{i,2}$ and from $u_{i,3}$ to $u_{i,4}$ can be interdicted at a removal cost of $N := \mu + 1$. Interdicting the arc from $u_{i,1}$ to $u_{i,2}$ is later identified with setting x_i to true and interdicting the arc from $u_{i,3}$ to $u_{i,4}$ is identified with setting x_i to false. For each clause $c_j \in C$, there is a *clause gadget* consisting of a directed path with trace $(v_{j,1}, v_{j,2}, v_{j,3}, v_{j,4})$ where only the arcs from $v_{j,1}$ to $v_{j,2}$ and from $v_{j,3}$ to $v_{j,4}$ can be interdicted at a removal cost of 1.

We next describe the arcs that connect the variable gadgets to the clause gadgets. For a clause $c_j = \hat{x}_i \vee \hat{x}_k$, where $\hat{x}_i \in \{x_i, \overline{x}_i\}$ and $\hat{x}_k \in \{x_k, \overline{x}_k\}$, we call $\hat{x}_i$ the *first literal* and $\hat{x}_k$ the *second literal* of clause c_j. For each clause, arcs are then added as follows depending on the clause's first and second literal: If the first literal of clause c_j is x_i ($\overline{x}_i$), there exists an arc from $u_{i,2}$ to $v_{j,1}$ (from $u_{i,4}$ to $v_{j,1}$). If the second literal of clause c_j is x_k ($\overline{x}_k$), there exists an arc from $u_{k,2}$ to $v_{j,3}$ (from $u_{k,4}$ to $v_{j,3}$).

The construction is continued by adding another six nodes s_1, s_2, s_3, t_1, t_2, and t_3 to the graph. For each $i \in \{1, \dots, \zeta\}$, there exists an arc from s_1 to $u_{i,1}$ and an arc from $u_{i,4}$ to t_1. For each $i \in \{1, \dots, \zeta\}$, there exists an arc from s_2 to $u_{i,1}$ and another arc from s_2 to $u_{i,3}$. Further,

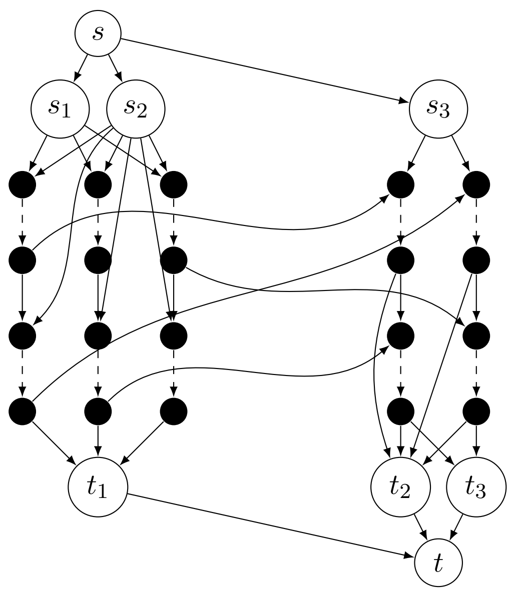


Figure 5.1.: The constructed graph for $X = \{x_1, x_2, x_3\}$ and $C = \{x_1 \vee \overline{x}_2, \overline{x}_1 \vee x_3\}$. The three variable gadgets for x_1, x_2, and x_3 from left to right are shown on the left and the clause gadgets for c_1 and c_2 from left to right are shown on the right. Only the dashed arcs can be interdicted.

for each $j \in \{1, \dots, \mu\}$, there exists an arc from $v_{j,2}$ to t_2 and from $v_{j,4}$ to t_2. Finally, for each $j \in \{1, \dots, \mu\}$, there exists an arc from s_3 to $v_{j,1}$ and an arc from $v_{j,4}$ to t_3.

We finish the construction by adding the nodes s and t to the graph. For each $k \in \{1, 2, 3\}$, there exists an arc from s to s_k with start time $1-k$ and traversal time $k-1$, and an arc from t_k to t with traversal time $3-k$ (but start time 0). As usual, the constructed temporal graph is denoted by $G = (V, R)$ in the following.

The budget is chosen to be $B := N \cdot \zeta + 2 \cdot \mu - \delta$, which completes the construction of the problem instance.

We show strong $\mathcal{NP}$-hardness by proving that there exists a truth assignment for the variables that satisfies at least δ clauses in the instance of MAX2SAT if and only if there exists a solution for the constructed instance with objective value at least 3. To this end, the following auxiliary result is required.

Lemma 5.13 Let D be an interdiction strategy in the constructed instance. The objective value of D is larger than or equal to 3 if and only if the pairs (s_1, t_1), (s_2, t_2), and (s_3, t_3) are separated by D.

Proof. If one of the pairs (s_1, t_1), (s_2, t_2), or (s_3, t_3) is not separated by D, it follows immediately that, after interdiction, there exists an s-t-path with duration 2. Hence, the solution D has objective value at most 2. To show the other direction, assume that the objective value of the solution is strictly less than 3 and let P_{SD} be a shortest duration path in G_D. If P_{SD} visits both s_k and t_k for some $k \in \{1,2,3\}$, we are done. If this is not the case, there are three possible pairs of nodes, one of which must be visited by P_{SD} since its duration is strictly less than 3 and every temporal s-t-path in G_D must visit one of the s_k and one of the t_k.

<u>Case 1:</u> P_{SD} visits s_1 and t_2

This means there exists a subpath P of P_{SD} from s_1 to t_2. The first arc in P leads into one of the nodes $u_{i,1}$ for some $i \in \{1,\dots,\zeta\}$. By replacing the first arc in P with the arc starting in s_2 and ending in $u_{i,1}$, we obtain a path from s_2 to t_2.

<u>Case 2:</u> P_{SD} visits s_2 and t_3

This means there exists a subpath P of P_{SD} from s_2 to t_3. The last arc in P starts from one of the nodes $v_{j,4}$ for some $j \in \{1,\dots,\mu\}$. By replacing the last arc in P with the arc starting in $v_{j,4}$ and ending in t_2, we again obtain a path from s_2 to t_2.

<u>Case 3:</u> P_{SD} visits s_1 and t_3

The first and the last arc in the subpath of P_{SD} from s_1 to t_3 can be replaced as in the previous two cases, which again yields a path from s_2 to t_2. ☐

Lemma 5.13 allows proving strong $\mathcal{NP}$-hardness of T-SDP-IP and T-STP-IP.

Theorem 5.14 T-SDP-IP and T-STP-IP are strongly $\mathcal{NP}$-hard even on acyclic graphs.

Proof. We show that there exists a truth assignment for the variables that satisfies at least δ clauses in the instance of MAX2SAT if and only if there exists a solution for the constructed T-SDP-IP instance with objective value at least 3.

First, let x be a truth assignment that satisfies at least δ clauses. We construct an interdiction strategy for the instance of T-SDP-IP with objective value at least 3 as follows. For each $i \in \{1,\dots,\zeta\}$, we interdict the arc from $u_{i,1}$ to $u_{i,2}$ if x_i is true and the arc from $u_{i,3}$ to $u_{i,4}$ if x_i is false. For each $j \in \{1,\dots,\mu\}$, we interdict the arc from $v_{j,1}$ to $v_{j,2}$ if the second literal in clause c_j is fulfilled, the arc from $v_{j,3}$ to $v_{j,4}$ if the second literal of c_j is not fulfilled, but the first is, and both of these arcs if none of the literals are fulfilled. This yields an interdiction strategy D that interdicts ζ arcs of cost N and at most $2 \cdot \mu - \delta$ arcs of cost 1 and, hence, does not exceed the budget.

Due to Lemma 5.13, it remains to show that the pair (s_k, t_k) is separated by D for each $k \in \{1,2,3\}$. Any path from s_1 to t_1 has trace $(s_1, u_{i,1}, u_{i,2}, u_{i,3}, u_{i,4}, t_1)$ for some $i \in \{1,\dots,\zeta\}$. As either the arc from $u_{i,1}$ to $u_{i,2}$ or the arc from $u_{i,3}$ to $u_{i,4}$ is interdicted, the pair (s_1,t_1) is separated by D. Moreover, the analogous argument applied to the clause gadgets shows that the pair (s_3,t_3) is separated by D.

To show that the pair (s_2,t_2) is separated by D, note that each path from s_2 to t_2 contains a subpath with trace $(u_{i,a}, u_{i,a+1}, v_{j,b}, v_{j,b+1})$ where $i \in \{1,\dots,\zeta\}$, $j \in \{1,\dots,\mu\}$, and $a, b \in \{1,3\}$. We interdict either the arc from $u_{i,a}$ to $u_{i,a+1}$ if the $(b+1/2)$-th literal of clause c_j is fulfilled or the arc from $v_{j,b}$ to $v_{j,b+1}$ if it is not. Hence, the pair (s_2,t_2) is separated by D and the objective value of D is at least 3 due to Lemma 5.13.

For the inverse direction, let $D \subseteq R$ be an interdiction strategy with objective value at least 3. In particular, this interdiction strategy removes arcs of total cost at most $B = N \cdot \zeta + 2 \cdot \mu - \delta$. Lemma 5.13 then implies that each pair (s_k,t_k), for $k \in \{1,2,3\}$, is separated.

In order to separate the pair (s_1, t_1), one arc has to be removed per variable gadget. For the sake of a contradiction, suppose that more than one arc is removed in some variable gadget. The total removal cost of interdicted arcs in the variable gadgets is then at least $N \cdot \zeta + N = N \cdot \zeta + \mu + 1$, which leaves only a budget of $\mu - \delta - 1 < \mu$ for interdicting arcs in the clause gadgets. Hence, there exists at least one clause gadget in which none of the arcs is interdicted. This implies that the pair (s_3, t_3) is not separated by D, which yields the desired contradiction. Overall, this means that, for each $i \in \{1, \ldots, \zeta\}$, either the arc from $u_{i,1}$ to $u_{i,2}$ is interdicted, in which case we set x_i to true, or the arc from $u_{i,3}$ to $u_{i,4}$ is interdicted, in which case we set x_i to false.

It remains to show that the resulting truth assignment fulfills at least δ clauses. As interdicting one arc per variable gadget already costs $N \cdot \zeta$, there is a budget of $2 \cdot \mu - \delta$ left for interdicting arcs in the clause gadgets. In order to separate the pair (s_3, t_3), at least one of the two removable arcs must be removed in each clause gadget. Hence, there are at least δ clause gadgets in which only one of the arcs is removed. We finish the proof by showing that x fulfills all the corresponding clauses c_j.

To this end, we first assume that the arc from $v_{j,3}$ to $v_{j,4}$ is interdicted. If the first literal in c_j is x_i, then there exists a path in G with trace $(s_2, u_{i,1}, u_{i,2}, v_{j,1}, v_{j,2}, t_2)$. As the arc from $v_{j,1}$ to $v_{j,2}$ is not interdicted and the pair (s_2, t_2) must be separated by D, this means that the arc from $u_{i,1}$ to $u_{i,2}$ must be interdicted and, hence, that x_i is set to true, which shows that x fulfills c_j. If the first literal in c_j is $\overline{x}_i$, then the same arguments hold for the path in G with trace $(s_2, u_{i,3}, u_{i,4}, v_{j,1}, v_{j,2}, t_2)$. The proof for the case when the arc from $v_{j,1}$ to $v_{j,2}$ is interdicted is along the same lines. Hence, at least δ clauses are fulfilled by x, which completes the proof. $\square$

Since any solution of the constructed T-SDP-IP instance that does *not* have objective value at least 3 has objective value at most 2, the proof of Theorem 5.14 further implies the following inapproximability result.

Corollary 5.15 Unless $\mathcal{P} = \mathcal{NP}$, there exists no polynomial-time approximation algorithm with approximation ratio smaller than $3/2$ for T-SDP-IP or T-STP-IP, even on acyclic graphs.

In the case of T-SDP-IP, the constructed instance in the reduction can easily be adjusted such that all traversal times are zero. To do so, all traversal times of the outgoing arcs of s are set to 0 and, for each incoming arc of t, the start time is increased by its traversal time and the traversal time is then set to 0. Hence, the results on T-SDP-IP from Theorem 5.14 and Corollary 5.15 are also valid for the definition of temporal graphs used in [Mol20].

In the case of T-STP-IP, however, using nonzero traversal times within the reduction is necessary. Indeed, the results on T-STP-IP from Theorem 5.14 and Corollary 5.15 do *not* hold for the definition in [Mol20] (unless $\mathcal{P} = \mathcal{NP}$) since T-STP-IP is solvable in polynomial time if all traversal times are zero as it then reduces to the question whether s can be separated from t by an interdiction strategy. It is, however, questionable, whether T-STP-IP has a meaningful interpretation in this case.

We continue by showing that the results of Theorem 5.14 and Corollary 5.15 (with a slight modification of the approximation ratio) also hold for instances with unit removal costs and strictly positive traversal times.

The restriction to unit removal costs can be achieved by replacing each arc r in the constructed graph by $c(r)$ identical copies with unit removal cost. Any interdiction strategy can then be assumed to either remove all of these identical copies or none of them. Moreover, since all removal costs are polynomial in the numbers of variables and clauses of the given MAX2SAT instance, the constructed instance with unit removal costs is still of polynomial size, so the arguments in the proof carry over to this instance.

For the restriction to strictly positive traversal times, note that the constructed graph G is acyclic. In particular, the graph $G - \{s,t\}$ is acyclic. Let $\sigma : V \to \{1, \ldots, n-2\}$ be a topological sorting of the nodes in $G - \{s,t\}$, i.e., for each arc r, it holds that $\sigma(\alpha(r)) < \sigma(\omega(r))$. This topological sorting is used to slightly modify the start and traversal times of the arcs in $G - \{s,t\}$. Formally, a function $\bar{\sigma} : V \to \{1, \ldots, n-2\}$ is constructed from the topological sorting by setting $\bar{\sigma}(v) := \sigma(v)$ if $v \notin \{s_1, s_2, s_3, t_1, t_2, t_3\}$, and $\bar{\sigma}(s_i) := 1$ and $\bar{\sigma}(t_i) := n-2$ for each $i \in \{1,2,3\}$. We then redefine the start and traversal times in G. To this end, let $\varepsilon \in (0,1)$. For each arc r that is not incident to s or t, we set the start time to $-{}^{\varepsilon}\!/_{2\cdot(n-3)} \cdot (n - 2 - \bar{\sigma}(\alpha(r)))$ and the traversal time to ${}^{\varepsilon}\!/_{2\cdot(n-3)} \cdot (\bar{\sigma}(\omega(r)) - \bar{\sigma}(\alpha(r)))$, which means that it arrives in $\omega(r)$ at time $-{}^{\varepsilon}\!/_{2\cdot(n-3)} \cdot (n-2-\bar{\sigma}(\omega(r)))$. We further set the start time of the arc from s to s_1 to $-\varepsilon$ and its traversal time to ${}^{\varepsilon}\!/_{2}$, and we decrease the traversal times of the other two outgoing arcs of s by ${}^{\varepsilon}\!/_{2}$. Hence, all outgoing arcs of s have arrival time $-{}^{\varepsilon}\!/_{2}$. Moreover, we set the traversal time of the arc from t_3 to t to ε.

The proof of Theorem 5.14 for this new instance is along the same lines as before and the statement of Corollary 5.15 must be slightly changed (see Corollary 5.16). Note that the graph with the updated start and traversal times does not admit waiting in any node except for s and t as, for any node $v \in V \backslash \{s,t\}$, all incoming arcs arrive and all outgoing arcs start at time $-{}^{\varepsilon}\!/_{2\cdot(n-3)} \cdot (n-2-\bar{\sigma}(v))$. The result, hence, holds for both problems T-SDP-IP and T-STP-IP.

Thus, when strictly positive traversal times on all arcs are additionally assumed, Theorem 5.14 still holds, but Corollary 5.15 has to be adapted as follows:

Corollary 5.16 Unless $\mathcal{P} = \mathcal{NP}$, there exists no polynomial-time approximation algorithm with approximation ratio smaller than $({}^{3}\!/_{2}+\varepsilon)$ for T-SDP-IP and T-STP-IP on acyclic graphs with positive traversal times for any $\varepsilon > 0$.

5.3.3.1 Polynomial-Time Solvability on Extension-Parallel Graphs

In this section, we show that T-SDP-IP, T-STP-IP, and the static version S-SP-IP are polynomial-time solvable on extension-parallel (temporal) graphs.

A temporal graph consisting of two nodes s and t, and a single temporal arc from s to t is called a *temporal one-arc graph.* A temporal graph with two distinguished vertices s (the source) and t (the sink) is *series-parallel* if it is obtained from a set of temporal one-arc graphs by a finite sequence of series compositions (identifying the sink of the first graph with the source of the second graph) and parallel compositions (identifying the sources of the two graphs and identifying the sinks of the two graphs). If, further, for every series composition, one of the two composed graphs is a temporal one-arc graph, the graph is called *extension-parallel.* The definitions of series- and extension-parallel *static* graphs is completely analogous and can be found, e.g., in [EFM07].

The *decomposition tree* T_G of a series-parallel (temporal) graph G is a binary tree, where the leaves represent the arcs in the graph and the inner nodes labeled by S (series composition) or P (parallel composition) represent the types of compositions used to construct the graph. The decomposition tree can be computed in linear time [VTL82] and it can easily be seen that a series-parallel (temporal) graph is extension-parallel if and only if every inner node of T_G that is labeled by S has one child that is a leaf of T_G.
The following property of extension-parallel static graphs is used in our algorithm.

Lemma 5.17 Let $G = (V, R)$ be an extension-parallel static graph. Then there exists a subset $\bar{R} \subseteq R$ of arcs such that

1. each s-t-path in G contains exactly one arc from $\bar{R}$, and
2. each arc $r \in \bar{R}$ is contained in exactly one s-t-path P_r in G.

Proof. We present an algorithm that constructs $\bar{R}$ and a corresponding s-t-path P_r for each $r \in \bar{R}$. Note that, since extension-parallel graphs are always acyclic, the path P_r is uniquely determined by the set of arcs it traverses. Hence, we slightly abuse notation and identify each path P_r with the corresponding set of arcs it traverses. The idea of the algorithm is to process the nodes in the decomposition tree starting at the leaves by iteratively joining two already processed components of the graph until we reach the root node and obtain the final set $\bar{R}$.

Initially, we set $\bar{R} := R$ and $P_r := \{r\}$ for each $r \in R$ and mark all leaf nodes in the decomposition tree as processed. While not all nodes in the decomposition tree are marked as processed, we take an unprocessed (inner) node v in the decomposition tree whose two children have both been processed. If v is labeled by P, we simply mark v as processed while changing neither the set $\bar{R}$ nor any of the paths P_r, for $r \in \bar{R}$. If v is labeled by S, at least one of its children must be a leaf corresponding to an arc r. If only one of the children is a leaf node, then we remove r from $\bar{R}$ and delete P_r. Further, we add r to all paths $P_{r'}$ for which the leaf node that corresponds to r' is a successor of the non-leaf child of v in the decomposition tree. If both children of v are leaves, then the arc r that corresponds to its right child is removed from $\bar{R}$, the path P_r is deleted, and r is added to the path $P_{r'}$, where r' is the arc that corresponds to the left child of v. We then mark v as processed and proceed.

To show the correctness of the algorithm, note that each node v in the decomposition tree can be associated with the subgraph G_v of G whose arc set consists of those arcs that correspond to leaf nodes in the decomposition tree that are successors of v.

We claim that, after each iteration, for each processed node v that either has no parent (i.e, v is the root node) or whose parent is still unprocessed, it holds that $\bar{R}$ restricted to the arc set of G_v fulfills the properties from the lemma for G_v.

This is clearly the case when only the leaves have been processed since every subgraph G_v is then a one-arc graph. Now assume that the claim holds at the beginning of an iteration and let v be the node in the decomposition tree that is processed in the iteration. If v is labeled by P, each s-t-path in G_v is either completely contained in the graph associated with the left child of v in the decomposition tree or in the graph associated with the right child. Hence, since $\bar{R}$ and the paths P_r, $r \in \bar{R}$, are left unchanged, the claim also holds after processing v. If the processed node v is labeled by S and both its children are leaves, the claim clearly remains true. If the processed node v is labeled by S and only one of its children is a leaf, then this series composition corresponds to prepending or appending an additional arc to the graph G_w, where w denotes the non-leaf child of v. Hence, after the series composition, each path in G_w is extended by the arc r that corresponds to the leaf child of v, which is precisely what the algorithm does. Moreover, r is removed from $\bar{R}$ and P_r is deleted, which ensures that uniqueness is preserved in both properties from the lemma. Hence, the claim also holds after the iteration, which completes the proof. $\square$

Note that the proof of Lemma 5.17 is constructive and the set $\bar{R}$ together with the paths P_r for $r \in \bar{R}$ can be obtained in $\mathcal{O}(m^2)$ time. Indeed, this running time is asymptotically best-possible since already the space to store the paths P_r for $r \in \bar{R}$ is in $\mathcal{O}(m^2)$. In the following, given an extension-parallel temporal graph, we let $\bar{R}$ denote a subset of arcs that satisfies the properties of Lemma 5.17 in the underlying static graph. For each arc $r \in \bar{R}$, we remove r from the graph and from $\bar{R}$ if the corresponding s-t-path P_r in the underlying static graph is not a (temporal) s-t-path in the temporal graph. Note that this does not destroy any temporal s-t-paths.

The idea of the polynomial-time algorithm to solve T-SDP-IP, T-STP-IP, and the static version S-SP-IP is similar to the idea of the algorithm for T-LSP-IP from Section 5.3.1. For ease of notation, the following exposition is restricted to T-SDP-IP. It is discussed later how the arguments can be modified for the other two problems.

Let $\text{dura}_1 < \text{dura}_2 < \cdots < \text{dura}_l$ denote the distinct durations of s-t-paths in G sorted in increasing order. Further, for $k \in \{1, \dots, l\}$, we define $G^{\text{SD},k}$ as the temporal graph that results from G by removing each arc $r \in \bar{R}$ for which P_r has duration at least dura_k. For completeness, we also define $G^{\text{SD},l+1} := G$. The following proposition and its proof are similar to Proposition 5.5 and the corresponding proof.

Proposition 5.18 Let $k \in \{1, \dots, l\}$. There exists an interdiction strategy D^k that separates s from t in $G^{\text{SD},k}$ if and only if there exists an interdiction strategy D in G with objective value at least dura_k.

Proof. Let D^k be an interdiction strategy that separates s from t in $G^{\text{SD},k}$. Then, after interdicting the same set $D := D^k$ of arcs in G, no s-t-path P in G_D can have duration less than dura_k (otherwise, P would also be an s-t-path in $G^{\text{SD},k}_{D^k}$ as (1) no arc in P is in $D = D^k$, and (2) the unique arc r in P contained in $\bar{R}$ satisfies $P_r = P$ and, thus, $\text{dura}(P_r) = \text{dura}(P) < \text{dura}_k$). Hence, all s-t-paths in G_D have duration at least dura_k, i.e., D has objective value at least dura_k.

Conversely, let D be an interdiction strategy in G with objective value at least dura_k. Then, no s-t-path in G_D can have duration less than dura_k, so the interdiction strategy $D^k := D \backslash \{r \in \bar{R} \mid \text{dura}(P_r) \geqslant \text{dura}_k\}$ separates s from t in $G^{\text{SD},k}$. $\square$

As in the algorithm presented in Section 5.3.1, the idea of the algorithm for extension-parallel (temporal) graphs is to use binary search in order to find $k^\star \in \{1, \dots, l\}$ such that s can be separated from t in $G^{\text{SD},k^\star}$, but s cannot be separated from t in $G^{\text{SD},k^\star+1}$. Such a $k^\star$ exists whenever s cannot already be separated from t in the whole graph $G = G^{\text{SD},l+1}$, i.e., whenever the optimal objective value is not equal to $+\infty$.

As shown in Section 5.3.1, deciding whether a node s can be separated from another node t with a given interdiction budget in an arbitrary temporal graph is possible in polynomial time. Further, Lemma 5.17 implies that the total number of s-t-paths is bounded by the number of arcs in the graph. Consequently, the number l of distinct durations of s-t-paths is polynomial in the input size. Altogether, the proposed algorithm runs in polynomial time.

To extend the result to the problems T-STP-IP and S-SP-IP, one observes the distinct traversal times or lengths of s-t-paths, respectively, to construct the subgraphs used in the algorithm. All arguments then work along the same lines.
The following theorem summarizes the main results of this section.

Theorem 5.19 There exist polynomial-time algorithms for T-SDP-IP, T-STP-IP, and S-SP-IP on extension-parallel (temporal) graphs with running time in $\mathcal{O}(m^2 + \log(m) \cdot T_{\text{MC}}(n \cdot m, n \cdot m))$ for the temporal versions and running time in $\mathcal{O}(m^2 + \log(m) \cdot T_{\text{MC}}(n, m))$ for the static version, where $T_{\text{MC}}(\bar{n}, \bar{m})$ is the time required to compute a minimum s-t-cut in a static graph with $\bar{n}$ nodes and $\bar{m}$ arcs.

5.4 Extensions

In this section, we study three extensions of the temporal shortest path interdiction problem. The first extension is to allow for negative traversal times and it is shown that the results of Section 5.3 remain valid for T-LSP-IP, T-EAP-IP, and T-SDP-IP if paths are explicitly not required to be elementary. The second extension is motivated by [Cas+12] and allows for continuous-time availability of arcs. It is shown that even a slight generalization makes it hard to decide whether the nodes s and t can be separated by an interdiction strategy in a temporal graph. Finally, the third extension, motivated by [Cas+21], imposes an additional constraint on the maximum waiting time in a node. It is shown that the additional constraint does not change the results from Section 5.3.

5.4.1 Negative Traversal Times

In this section, we show that the results obtained for the problems T-LSP-IP, T-EAP-IP, and T-SDP-IP in Section 5.3 still hold if negative traversal times are allowed. For this extension, paths are explicitly not required to be elementary.

We start by arguing that the underlying shortest path problem for the three variants can be solved efficiently using the time-expanded graph. Note that, given a temporal graph $H = (V, R)$, the construction of the time-expanded graph $H^{\text{te}} = (V^{\text{te}}, R^{\text{te}})$ does not change when involving negative traversal times. The main difference that arises from allowing negative traversal times is that the time-expanded graph might have arcs from a node (v, ϕ_i) to a node (w, ϕ_k) with $\phi_k < \phi_i$. For the latest start path problem, the objective of a path only depends on the first arc in the path. It is easy to see that, for ϕ_j being the maximum crucial time, there exists an (s, ϕ_i)-(t, ϕ_j)-path in H^{te} if and only if there exists an s-t-path in H with start time at least ϕ_i. Note that this only holds if the path is not required to be elementary. By evaluating whether there exists an (s, ϕ_i)-(t, ϕ_j)-path in H^{te} for each crucial time ϕ_i, the latest start path problem can be solved in polynomial time when involving negative traversal times. A similar argumentation yields that the earliest arrival path problem and the shortest duration path problem can also be solved in polynomial time.

By the same argument as above, it is clear that the statement of Observation 5.6 still holds if negative traversal times are allowed. Note that, while allowing the traversal times to be negative, the removal costs are still positive and, hence, the problem of deciding whether s^{te} and t^{te} can be separated in $H^{\text{te}} = (V^{\text{te}}, R^{\text{te}})$ can be solved in polynomial time. Hence, T-LSP-IP can be solved in polynomial time if negative traversal times are allowed.

For the polynomial time solvability of T-EAP-IP, note that neither the construction of G^{LS} nor the proofs of Proposition 5.8 and Corollary 5.9 require the traversal times to be non-negative. Hence, since T-LSP-IP can be solved in polynomial time if negative traversal times are allowed, so can be T-EAP-IP.

Finally, for T-SDP-IP, it is evident that allowing for negative traversal times is a generalization of the problem studied in Section 5.3 and, hence, the same proof of hardness shows the hardness in the generalized setting. Since the underlying shortest duration path problem can be solved in polynomial time, T-SDP-IP is still $\mathcal{NP}$-hard when allowing negative traversal times.

The results of this section are summarized in the following theorem:

Theorem 5.20 When allowing negative traversal times, T-LSP-IP and T-EAP-IP are polynomial-time solvable, while T-SDP-IP is $\mathcal{NP}$-hard.

For T-STP-IP, the proof of hardness given in Section 5.3.3 also holds true for the same reasons as for T-SDP-IP. However, it is per se not clear whether the problem is in $\mathcal{NP}$ since it is not evident that a shortest traversal path in a temporal graph can still be computed in polynomial time when negative traversal times are allowed.

5.4.2 Continuous Time Availability of Arcs

In this section, a slightly more general model of temporal graphs is investigated, where the start time $\tau(r)$ of a temporal arc r is not given by one fixed discrete point in time, but rather by a closed interval $[\tau^{\text{l}}(r), \tau^{\text{u}}(r)] =: \tau(r)$. The arc can then be entered at any time $\tau \in \tau(r)$, leading to an arrival time of $\tau + \lambda(r)$ at $\omega(r)$. In the following, we therefore no longer speak of a start time, but of an *availability interval.* The resulting temporal graphs where arcs are available during availability intervals are called *continuous-time temporal graphs.* Note that the class of continuous-time temporal graphs comprises the class of temporal graphs, which correspond to continuous-time temporal graphs in which each availability interval only consists of a single point.

While a temporal path in a discrete-time temporal graph is given by a sequence of temporal arcs, the definition has to be slightly adapted in the continuous-time case. A *(continuous-time) temporal path* in a continuous-time temporal graph is a sequence $P = ((r_1, \tau_1), \ldots, (r_k, \tau_k))$ of pairs of an arc and a start time with $\tau_i \in \tau(r_i)$ for each $i \in \{1, \ldots, k\}$ such that $\omega(r_i) = \alpha(r_{i+1})$ and $\tau_i + \lambda(r_i) \leqslant \tau_{i+1}$ for each $i \in \{1, \ldots, k-1\}$.

We show that, given a continuous-time temporal graph G and an integer B, it is strongly $\mathcal{NP}$-hard to decide whether a pair (s, t) of nodes can be separated by removing at most B

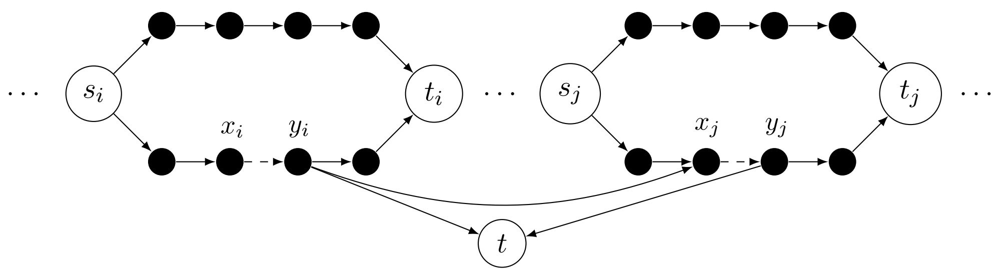


Figure 5.2.: The gadgets in G for two nodes i and j that are adjacent in G' (where $i < j$). Only the dashed arcs can be interdicted.

arcs from G. This immediately implies that all variants of temporal shortest path interdiction problems studied in this chapter are strongly $\mathcal{NP}$-hard on continuous-time temporal graphs, and that they do not not even admit polynomial-time approximation algorithms with a bounded approximation ratio (unless $\mathcal{P} = \mathcal{NP}$).

The reduction, which is similar to the one presented in [BKS95], is from the well-known (strongly) $\mathcal{NP}$-hard node cover problem, which is defined as follows:

Node Cover

INSTANCE:	An undirected (static) graph $G' = (V', E)$ and a positive integer $B' \leqslant \lvert V' \rvert$
QUESTION:	Is there a subset $\bar{V} \subseteq V'$ of nodes with $\lvert \bar{V} \rvert \leqslant B'$ such that each edge in E is incident to at least one node in $\bar{V}$?

Given an instance of node cover, a continuous-time temporal graph G and a budget B are constructed as follows. The budget is chosen as $B := B'$. For the construction of the continuous-time temporal graph G, it is assumed without loss of generality that $V' = \{1, \ldots, n'\}$. For each node $i \in \{1, \ldots, n'\}$, a gadget consisting of two parallel paths of length five is constructed. These paths are referred to as the *upper* and *lower* path of the gadget. The start node of the two paths is referred to as s_i and the end node of the two paths as t_i. Further, the start and end node of the third arc in the lower path are referred to as x_i and y_i, respectively. All arcs in a gadget, except for the third arc in the lower path, have removal cost $B + 1$ and availability interval $[0, 5n']$. The third arc in the lower path has removal cost 1 and availability interval $[5i - 4, 5i - 3]$. All arcs in the gadget have traversal time 1.

The gadgets are connected by identifying t_i with s_{i+1} for all $i \in \{1, \ldots, n' - 1\}$. Further, for an edge $e \in E$ that is incident to the nodes i and j with $i < j$, there exists an arc from y_i to x_j with availability interval $[5i - 2, 5i - 2]$, traversal time $5(j - i) - 2$, and removal cost $B + 1$. This arc is called the *shortcut* from i to j.

The node s for the instance of temporal shortest path interdiction is s_1. The node t is added and, for each $i \in \{1, \ldots, n'\}$, there exists an arc from y_i to t with availability interval $[5i - 3, 5i - 3]$, traversal time 0, and removal cost $B + 1$. An illustration of the construction is provided in Figure 5.2.

To achieve unit removal costs, we can simply replace each arc r by $c(r)$ many identical copies with unit removal cost (as for the proof of Theorem 5.14 and Corollary 5.15 for unit removal costs). Note that this conserves the polynomial size of the constructed instance. Using this construction, we prove the following theorem:

Theorem 5.22 Deciding whether a pair (s, t) of nodes can be separated by removing at most B arcs from a continuous-time temporal graph is strongly $\mathcal{NP}$-complete.

Proof. The problem is clearly in $\mathcal{NP}$ since it can be easily checked in polynomial time whether a given set of at most B arcs separates s from t. To show $\mathcal{NP}$-completeness, let D be an interdiction strategy for the constructed instance. Observe that each s-t-path in G_D traverses at least one shortcut from i to j for some $i, j \in \{1, \ldots, n'\}$. This is possible if and only if none of the removable arcs in gadgets i and j are removed by the interdiction strategy. By identifying the interdiction of an interdictable arc in a gadget i with the inclusion of node i in the node cover, it follows that there exists a node cover of size B' $(= B)$ if and only if there exists an interdiction strategy in G separating s from t that removes at most B arcs. $\square$

5.4.3 Waiting Time Constraints

The definition of an s-t-path provided in Section 5.2 implicitly allows to wait at nodes for any length of time. However, arbitrarily long waiting times are often undesired in real-world problems such as, e.g., packet routing in communication networks. To this end, the problem of finding a Δ-*restless temporal* s-t-*path* that cannot wait longer than a given amount of time Δ in any node except s and t has been investigated (see [Cas+21]). As shown in [Cas+21], deciding whether an *elementary* Δ-restless s-t-path exists is strongly $\mathcal{NP}$-hard for any $\Delta \geqslant 0$.[3] However, in the setting considered here where paths are not required to be elementary, this problem is polynomial-time solvable. A Dijkstra-like polynomial-time algorithm for computing *not necessarily elementary* restless paths in temporal graphs is presented in [Ben+20].

In this section, we show how the time-expanded graph introduced in Section 5.3.1 can be modified to account for additional waiting time constraints. Further, we show that the complexity of the four versions of temporal shortest path interdiction does not change under additional waiting time constraints.

Within this section, we assume that s has no incoming arcs and t has no outgoing arcs. This assumption does not impose a loss of generality since a shortest s-t-path (with respect to any of the definitions of "shortest") that uses such an arc could be transformed into one that does not.

Given an arbitrary temporal graph $H = (V_H, R_H)$, we construct the time-expanded graph $H^{\text{te}} = (V^{\text{te}}, R^{\text{te}})$ under waiting time constraints. To this end, recall the set $T = \{\phi_1, \ldots, \phi_j\}$ of crucial times, which are indexed in increasing order. Similar to the construction of the time-expanded graph in Section 5.3.1, we introduce a node (v, ϕ_i) for every $v \in V_H$ and $i \in \{1, \ldots, j\}$. For $v \in \{s, t\}$ and $i \in \{1, \ldots, j - 1\}$, there exists an arc from (v, ϕ_i) to (v, ϕ_{i+1}),

[3]It is worth noting that the definition of temporal graphs in [Cas+21] is slightly different. They state that the problem is strongly $\mathcal{NP}$-hard for any $\Delta \geqslant 1$, but, indeed, the same proof of hardness with a slight modification holds true for the definition of temporal graphs used here for any $\Delta \geqslant 0$.

which represents waiting at s before the start of the path or waiting at t after having arrived. For each arc $r \in R_H$, an additional node u_r is introduced. Further, there exists an arc in R^{te} from $(\alpha(r), \tau(r))$ to u_r and an arc from u_r to any node $(\omega(r), \phi)$ with $\phi \in [\tau(r) + \lambda(r), \tau(r) + \lambda(r) + \Delta]$. Traversing the arc from $(\alpha(r), \tau(r))$ to u_r and then the arc from u_r to some node $(\omega(r), \phi)$ represents traversing r in the temporal graph and entering the next arc in the path (if $\omega(r) \neq t$) exactly at time ϕ. This completes the construction of H^{te}.

To show a one to one correspondence between Δ-restless s-t-paths in H and (s, ϕ_1)-(t, ϕ_j)-paths in H^{te}, note that there are no parallel arcs in H^{te}, which implies that any path in H^{te} is uniquely given by its trace.

Observation 5.23 There exists a Δ-restless s-t-path in H if and only if there exists a (s, ϕ_1)-(t, ϕ_j)-path in H^{te}.

Proof. Let $P = (r_1, \ldots, r_k)$ be a Δ-restless s-t-path in H. Then we claim the unique path with trace $((s, \phi_1), \ldots, (s, \tau(r_1)), u_{r_1}, (\omega(r_1), \tau(r_2)), u_{r_2}, \ldots, (t, \tau(r_k) + \lambda(r_k)), \ldots, (t, \phi_j))$ is a (s, ϕ_1)-(t, ϕ_j) path in H^{te}. Since, for $v \in \{s, t\}$ and $i \in \{1, \ldots, j-1\}$, there exists an arc from (v, ϕ_i) to (v, ϕ_{i+1}), the arcs from (s, ϕ_1) to $(s, \tau(r_1))$ and the arcs from $(t, \tau(r_k) + \lambda(r_k))$ to (t, ϕ_j) are in H^{te}. Moreover, the arc from $(\alpha(r_i), \tau(r_i))$ to u_{r_i} is in H^{te} for $i \in \{1, \ldots, k\}$, and the arc from u_{r_i} to $(\omega(r_i), \tau(r_{i+1}))$ is in H^{te} for $i \in \{1, \ldots, k-1\}$ since P is Δ-restless.

Conversely let P' be an (s, ϕ_1)-(t, ϕ_j)-path in H^{te}. From the construction, it immediately follows that the trace of P' must be of the above form and, by the same arguments as above, it follows that $(r_1, \ldots, r_k)$ is a Δ-restless s-t-path in H. □

To show that T-LSP-IP and T-EAP-IP remain polynomial-time solvable, we assign removal costs to $H^{\text{te}} = (V^{\text{te}}, R^{\text{te}})$. For each arc $r \in R$, the (unique) incoming arc of u_r has removal cost $c(r)$. All other arcs have removal cost $B + 1$. With this construction, we observe the following.

Observation 5.24 There exists an interdiction strategy D such that there does not exist a Δ-restless path in H_D if and only if there exists an interdiction strategy D' separating s^{te} from t^{te} in H^{te}.

Using the algorithm proposed in Sections 5.3.1 and 5.3.2 together with the time-expanded graph $H^{\text{te}} = (V^{\text{te}}, R^{\text{te}})$ under waiting time constraints constructed in this section, it follows that T-LSP-IP and T-EAP-IP remain polynomial-time solvable under waiting time constraints.

Theorem 5.25 There exists a polynomial-time algorithm for solving T-LSP-IP and T-EAP-IP under waiting time constraints for each $\Delta \geqslant 0$.

We proceed with assessing the complexity of T-SDP-IP and T-STP-IP under waiting time constraints. When taking a closer look at the reduction provided in Section 5.3.3, every s-t-path in the constructed instance is 0-restless. This immediately implies that T-SDP-IP and T-STP-IP under waiting time constraints are strongly $\mathcal{NP}$-hard for every $\Delta \geqslant 0$, which yields the following theorem.

Theorem 5.26 The problems T-SDP-IP and T-STP-IP are strongly $\mathcal{NP}$-hard under waiting time constraints for each $\Delta \geqslant 0$.

To close this chapter, we argue that T-SDP-IP and T-STP-IP are still polynomial-time solvable on extension-parallel temporal graphs under waiting time constraints. To this end, when removing each arc r whose corresponding s-t-path P_r is not a temporal path from the graph and from the set $\bar{R}$ as in Lemma 5.17, we additionally check whether P_r is Δ-restless and remove r if this is not the case. Afterwards, the graph contains exactly the Δ-restless temporal s-t-paths and the algorithm presented in Section 5.3.3.1 can be used to solve T-SDP-IP and T-STP-IP on extension-parallel temporal graphs under waiting time constraints, which yields the following theorem.

Theorem 5.27 There exists a polynomial-time algorithm for T-SDP-IP and T-STP-IP on extension-parallel (temporal) graphs under waiting time constraints for each $\Delta \geqslant 0$.

It is worth noting that all results presented in this section can easily be adapted to the general case where waiting times are constrained node-wise instead of globally. The only difference to the global case is in the construction of the time-expanded graph, where the node-wise constriction is enforced by the outgoing arcs of the nodes u_r for $r \in R$.

5.5 Conclusion

In this chapter, the complexity of four different versions of temporal shortest path interdiction is analyzed. While the latest start and the earliest arrival path interdiction problem is shown to be solvable in polynomial time, the shortest duration and the shortest traversal path interdiction problem is strongly $\mathcal{NP}$-hard. It is particularly interesting that, even though *temporal* shortest path interdiction seems more complex than its static counterpart, which is known to be strongly $\mathcal{NP}$-hard, there are versions of temporal shortest path interdiction problems that are polynomially solvable. We further provide polynomial-time algorithms for the two hard problems on extension-parallel temporal graphs, which can also be transferred to the static shortest path interdiction problem.

An interesting direction for future work could be to study temporal shortest path interdiction problems for other types of modifications than arc removal. For example, one could consider the problem of worsening (or improving) the latest start time, the earliest arrival time, the shortest duration, or the shortest traversal time as much as possible by changing a given number of start times of arcs in a temporal graph.

6 Approximating Single- and Multi-Objective Nonlinear Sum and Product Knapsack Problems

Abstract We present an FPTAS for a very general version of the well-known knapsack problem. This generalization covers, with few exceptions, all versions of knapsack problems that have been studied in the literature so far and allows for an objective function consisting of sums or products of possibly nonlinear, separable item profits, while the knapsack constraint states an upper bound on the sum of possibly nonlinear, separable item weights. Moreover, we extend our FPTAS to a MFPTAS for the multi-objective version of the problem.
As applications of our general algorithms, we obtain the first FPTAS for the recently-introduced 0-1 time-bomb knapsack problem as well as FPTASs for a variety of robust knapsack problems. Moreover, we extend our FPTAS to the minimization version of our general problem, which, in particular, allows us to explicitly state an FPTAS for the classical minimization knapsack problem, which has been missing in the literature so far.

6.1 Introduction

Integer optimization problems with linear objectives and constraints are becoming more and more tractable through the availability of better computing hardware and the impressive progress of solver software (see [Koc+22]). This fostered the progress of research on nonlinear integer programs leading to a wealth of exact and heuristic solution algorithms for nonlinear optimization problems over the last two decades, as illustrated by the wide interest in the MIPLIB 2017 collection of benchmark instances [Gle+21].

We aim at complementing this development from a more theoretical perspective by presenting approximation schemes for a very general nonlinear version of the classical knapsack problem. This generalization consists of maximizing either the sum $\sum_{i=1}^{n} f_i(x_i)$ or the product $\prod_{i=1}^{n} f_i(x_i)$ of the profits $f_i(x_i)$ obtained from packing x_i copies of item $i \in \{1, \ldots, n\}$, where x_i is bounded by an item-specific upper bound u_i. Each number of copies x_i of item i

gives rise to a packed weight $g_i(x_i)$, where the total packed weight across all items is bounded by a knapsack capacity C. Both functions f_i and g_i can be arbitrary nonlinear functions.

The current literature on nonlinear knapsack problems mostly restricts the objective function to the sum of the profits $f_i(x_i)$, which significantly limits the applicability of the developed algorithms. As an example, a common way to represent preferences over commodity bundles in economics is to use Cobb-Douglas functions (see [MWG95]). In this setting, the input is given by discrete quantities of different consumption goods, which determine a utility value via a utility function of the form

$$\max \prod_{i=1}^{n} x_i^{\alpha_i}.$$

The optimal consumption bundle is then found by maximizing this utility function subject to a budget constraint. This clearly motivates the consideration of the product of the profits $f_i(x_i)$. Further pointers to literature about the relevance of the product in the objective function can be found in [DAm+18].

The standard knapsack problem can be seen as a cornerstone of integer programming and has been treated extensively with respect to every conceivable aspect (see [KPP04] and the recent literature survey [Cac+22a; Cac+22b]). However, there is only a fairly limited amount of literature on nonlinear versions of the knapsack problem. Moreover, these contributions are focused on an algorithm engineering perspective, while there are hardly any contributions on approximation algorithms, although these are widely studied for the linear case (most recently in [Jin19]).

In this chapter, we devise fully polynomial-time approximation schemes (FPTASs), which guarantee feasible solutions with strictly limited deviation from optimality for every given accuracy $\varepsilon > 0$ within a running time polynomial in the size of the instance and in $1/\varepsilon$. In contrast to many other approximation algorithms that are constructed only for their theoretical properties, the FPTASs presented in this chapter could be quite easily implemented in practice and would reach solutions with a given accuracy in a reasonable computation time.

We would like to point out that the notion of nonlinearity employed here encompasses basically every nonlinear knapsack problem with separable profit and weight functions. Profits contributed by different items can be combined either in an additive or in a multiplicative way, while weight values are summed up. This model is far more general than the few FPTASs known so far and will be applicable to every setting with separable functions.

Taking into account the manifold perspectives inherent in complex decision making tasks, multi-objective optimization models have become more widely used in practice and, therefore, also studied more intensively from an algorithmic point of view. Following this perspective, we extend our FPTAS to the multi-objective case, allowing a mix of additive and multiplicative objectives. This multi-objective setting is even more relevant for the following reason. It is well known that the set of efficient solutions (Pareto solutions) as well as the set of nondominated points (nondominated images) for multi-objective integer programming problems, such as the multi-objective knapsack problem, may have exponential size [Ehr05] and, thus, lead to an information overflow for a decision maker. By resorting to a multi-objective FPTAS, one can restrict the size of the solution set at the cost of a limited loss of accuracy, while still guaranteeing that all parts of the Pareto frontier are adequately represented in the final solution set (see, e.g., [HRT21]).

6.1.1 Previous Work

For a general overview of the literature about nonlinear knapsack problems, we refer to [BS02] and [Cac+22b, Sec. 5.1]. Concerning approximation, an FPTAS for a nonlinear version of the knapsack problem is presented in [Kov96]. They present a $(1 + \varepsilon)$-approximation algorithm for additive, nondecreasing profit functions f_i, and additive (although this is a bit hidden in the paper) nondecreasing weight functions g_i. The running time depends on the sum of the upper bounds u_i of item copies. If upper and lower bounds on the optimal objective value whose ratio is polynomial in the length of the encoded input are given, the algorithm becomes an FPTAS.

Another FPTAS for a yet more constrained nonlinear version of the knapsack problem is presented in [Hoc95]. They present an FPTAS with running time only depending on the number n of items (instead of the sum of the u_i) and $\log C$ for the case where the profit functions f_i are additive, concave, and nondecreasing, and the weight functions g_i are additive, convex, and nondecreasing.

A matheuristic for a nonlinear knapsack problem is given in [DM11]. This algorithm, however, does not yield any bounded approximation guarantee. Another metaheuristic for an even more complex scenario, in which there are multiple knapsacks the items can be assigned to, is examined in [DMM18].

The only knapsack-type problem with a multiplicative objective function treated in the literature so far is the product knapsack problem (PKP) introduced in [DAm+18], where a dynamic-programming algorithm with running time in $\mathcal{O}(nC)$ is presented. PKP considers binary variables x_i with constant profit values. An interesting aspect arises from allowing negative profits, which gives relevance to the parity of the number of packed items. A proof of weak $\mathcal{NP}$-hardness for PKP is given in [Hal+19] and an FPTAS is presented in [PST21].

The multi-objective version of the classical (linear, additive) knapsack problem has also been studied extensively in the literature and several MFPTASs are known. For an overview, we refer to [HRT21, Section 3.5].

6.1.2 Our Contribution

In this chapter, we present an FPTAS for a very general version of the well-known knapsack problem, where the objective function consists of sums or products of possibly nonlinear, separable item profits, while the knapsack constraint states an upper bound on the sum of possibly nonlinear, separable item weights. Besides the separability of profits and weights, the only assumption imposing a loss of generality in our problem definition is that all profits must be nonnegative. Hence, with few exceptions, our generalized version of the knapsack problem covers all versions of knapsack problems that have been studied in the literature, and our results answer the question about the existence of an FPTAS for any separable, nonlinear knapsack problem. Moreover, we extend our FPTAS to an MFPTAS for the multi-objective version of the problem.

We conclude the chapter by presenting applications of our general algorithms to various problems from the literature. In particular, we obtain the first FPTAS for the 0-1 time-bomb knapsack problem, which has recently been introduced in [MPS22], and FPTASs for a variety of robust knapsack problems. In addition, extending our FPTAS to the minimization version of our general problem allows us to explicitly state an FPTAS for the classical minimization knapsack problem, which has been missing in the literature so far.

6.2 Problem Definition

In this section, a highly generalized version of the knapsack problem is introduced in a single- and multi-objective setting.

6.2.1 Single-Objective Problems

We are given a set $\mathcal{N} = \{1, \ldots, n\}$ of n items, where each item $i \in \mathcal{N}$ can be packed at most $u_i \in \mathbb{N}_{>0}$ times. The obtained profit and the required capacity when packing $0 \leqslant x_i \leqslant u_i$ copies of item i are given by the values $f_i(x_i)$ and $g_i(x_i)$ of the nonnegative profit function $f_i : \{0, \ldots, u_i\} \ni x_i \mapsto f_i(x_i) \in \mathbb{Q}_{\geqslant 0}$ and the (arbitrary sign) weight function $g_i : \{0, \ldots, u_i\} \ni x_i \mapsto g_i(x_i) \in \mathbb{Q}$, respectively. Lower bounds on the number of item copies could also be included, in which case x_i would be restricted to $\ell_i \leqslant x_i \leqslant u_i$. Given a knapsack capacity $C \in \mathbb{Q}_{\geqslant 0}$, the task then consists of maximizing either the sum $\sum_{i=1}^n f_i(x_i)$ or the product $\prod_{i=1}^n f_i(x_i)$ of the profits obtained from the items while packing a total weight $\sum_{i=1}^n g_i(x_i)$ of at most C and packing at most u_i copies of each item i. The general problem can, thus, be formulated as follows:

$$\max \sum_{i=1}^n f_i(x_i) \quad \text{or} \quad \prod_{i=1}^n f_i(x_i) \tag{6.1}$$

$$\text{s.t.} \sum_{i=1}^n g_i(x_i) \leqslant C \tag{6.2}$$

$$0 \leqslant x_i \leqslant u_i \qquad \forall i \in \{1, \ldots, n\} \tag{6.3}$$

$$x_i \in \mathbb{Z} \qquad \forall i \in \{1, \ldots, n\} \tag{6.4}$$

In the following, we refer to the problem version with the sum objective $\sum_{i=1}^n f_i(x_i)$ as the *nonlinear additive knapsack problem* (NAKP) and to the problem version with the product objective $\prod_{i=1}^n f_i(x_i)$ as the *nonlinear product knapsack problem* (NPKP). Any vector $x \in \mathbb{Z}^n$ with $0 \leqslant x_i \leqslant u_i$ for all $i = 1, \ldots, n$ is called a *solution*. If such a solution additionally satisfies the capacity constraint $\sum_{i=1}^n g_i(x_i) \leqslant C$, it is called a *feasible solution*. The set of all feasible solutions is referred to as the *feasible set* and is denoted by $X \subseteq \mathbb{Z}^n$.
Throughout the chapter, we make the following assumptions.

Assumption 6.1 Any instance of NAKP or NPKP satisfies:

(a) All profits are nonnegative, i.e., $f_i(x_i) \geqslant 0$ for $0 \leqslant x_i \leqslant u_i$ and all $i \in \mathcal{N}$.

(b) The values $f_i(x_i)$ and $g_i(x_i)$ of the profit and weight functions are integers (i.e., $f_i(x_i), g_i(x_i) \in \mathbb{Z}$) that are given explicitly in the input for $0 \leqslant x_i \leqslant u_i$ and all $i \in \mathcal{N}$.

(c) All weights are nonnegative, i.e., $g_i(x_i) \geqslant 0$ for $0 \leqslant x_i \leqslant u_i$ and all $i \in \mathcal{N}$.

(d) There exists at least one feasible solution $x \in X$.

Note that Property (a) in Assumption 6.1 has only been listed for completeness since the profit functions have been assumed to be nonnegative already in the problem definition. Properties (b)–(d), on the other hand, can easily be seen to impose no real loss of generality. For Property (b), first note that the assumption of integer values can be ensured in the presence of rational values by multiplying all numbers by their lowest common denominator.[1] It should be noted that rational numbers can have an impact on the classification as weakly or strongly $\mathcal{NP}$-hard, as recently shown in [Woj18] for the knapsack problem. However, this does not affect the existence of an FPTAS (see [Woj18]). The important implication of Property (b) is that the values u_i are polynomial in the encoding length of a problem instance since $u_i + 1$ many values have to be encoded for f_i and g_i each. An alternative model would assume that each value $f_i(x_i)$ or $g_i(x_i)$ can be obtained in constant time via an oracle, in which case these values would not need to be encoded and the values u_i could be super-polynomial in the encoding length of a problem instance. In this case, however, even determining whether a feasible solution exists would require super-polynomial time in the worst case even in the special case of a single item (i.e., for $n = 1$) since – as the weight function g_1 is not assumed to be monotone – all the superpolynomially many solutions given by $x_1 = 0, \dots, u_1$ would have to be checked for feasibility.[2]

Concerning Property (c), we can determine $\min_{0 \leqslant x_i \leqslant u_i} g_i(x_i)$ for each item $i \in \mathcal{N}$ in $\mathcal{O}(u_i)$ time (which is polynomial in the encoding length of the input by Property (b)). Then, for each item i with $\min_{0 \leqslant x_i \leqslant u_i} g_i(x_i) < 0$, we can increase all weights of the item as well as the knapsack capacity by $-\min_{0 \leqslant x_i \leqslant u_i} g_i(x_i) > 0$ by setting $g_i(k) := g_i(k) - \min_{0 \leqslant x_i \leqslant u_i} g_i(x_i)$ for $k = 0, \dots, u_i$ and $C := C - \min_{0 \leqslant x_i \leqslant u_i} g_i(x_i)$, which yields an equivalent instance with $g_i(x_i) \geqslant 0$ for $0 \leqslant x_i \leqslant u_i$ and all $i \in \mathcal{N}$. Similarly, Property (d) imposes no loss of generality since packing $\operatorname{argmin}_{0 \leqslant x_i \leqslant u_i} g_i(x_i)$ copies of each item i clearly yields a feasible solution if one exists, so infeasible instances can be detected by preprocessing in polynomial time.

6.2.2 Multi-Objective Problems

In the *multi-objective nonlinear product-sum knapsack problem* (MNPSKP), we are given a set $\mathcal{N} = \{1, \dots, n\}$ of n items with nonlinear weight functions g_i and item-specific upper bounds

[1]Multiplying all values of the profit and weight functions as well as the knapsack capacity by their lowest common denominator $d \in \mathbb{Z}_{>0}$ does not impact feasibility of solutions, and the objective value of any solution is multiplied by the same positive factor d (in NAKP) or d^n (in NPKP).

[2]Even under the additional assumption that $g_i(x_i) \leqslant C$ for $0 \leqslant x_i \leqslant u_i$ and all i (which implies the existence of a feasible solution for $n = 1$), it is easy to see that no constant approximation ratio can be obtained in polynomial time in the oracle model since this would again require checking the superpolynomially many solutions given by $x_1 = 0, \dots, u_1$ in the worst case.

$u_i \in \mathbb{N}_{>0}$, for $i = 1, \dots, n$, and a knapsack capacity $C \in \mathbb{Q}_{\geqslant 0}$ as in the single-objective problems NAKP and NPKP. However, each item $i \in \mathcal{N}$ is now associated with a constant number $p \geqslant 2$ of profit functions $f_i^j : \{0, \dots, u_i\} \to \mathbb{Q}_{\geqslant 0}$, $j = 1, \dots, p$. The set $\{f^1, \dots, f^p\}$ of objectives to be maximized is partitioned by a threshold value $p' \in \{0, 1, \dots, p\}$ such that the objective functions $f^1, \dots, f^{p'}$ correspond to products, and the remaining objective functions $f^{p'+1}, \dots, f^p$ correspond to sums (where $p' = 0$ means that all objective functions are sums, while $p' = p$ means that all objective functions are products). Thus, the vector of objective values of any solution $x \in \mathbb{Z}^n$ can be written as:

$$f(x) = \left(\prod_{i=1}^{n} f_i^1(x_i), \dots, \prod_{i=1}^{n} f_i^{p'}(x_i), \sum_{i=1}^{n} f_i^{p'+1}(x_i), \dots, \sum_{i=1}^{n} f_i^{p}(x_i) \right) \tag{6.5}$$

Note that, in the following, we again assume as in Assumption 6.1 (b)–(d) that the values of the profit and weight functions are integers (i.e., $f_i^j(x_i), g_i(x_i) \in \mathbb{Z}$ for all i, j and $0 \leqslant x_i \leqslant u_i$) and are given explicitly in the input, that all weights are nonnegative (i.e., $g_i(x_i) \geqslant 0$ for all i and $0 \leqslant x_i \leqslant u_i$), and that at least one feasible solution $x \in X$ exists. This can again be seen to be without loss of generality by the same arguments as before.

For the special case of binary variables, where $u_i = 1$ for all $i \in \mathcal{N}$, we obtain linear profits and weights. In this case, for $p' = 0$, MNPSKP coincides with the classical multi-objective knapsack problem, which is widely studied in the literature. Moreover, for $p' = p$, we obtain the multi-objective generalization of the product knapsack problem (PKP) mentioned in the introduction. This extension of PKP to more than one objective has not been studied in the literature so far.

6.3 A Single-Objective FPTAS

In this section, we present an FPTAS that works for both NAKP and NPKP. In order to allow for a general formulation of the algorithm and its analysis that applies to both problems simultaneously, we use the symbols $\otimes \in \{+, \cdot\}$ and $\bigotimes \in \{\sum, \prod\}$ to denote either addition ($+$ and $\sum$) for the case of NAKP or multiplication ($\cdot$ and $\prod$) for the case of NPKP.

We first observe that the two values $\text{ub}_+ := \sum_{i=1}^n \max_{0 \leqslant x_i \leqslant u_i} f_i(x_i)$ and $\text{ub}_\cdot := \prod_{i=1}^n \max_{0 \leqslant x_i \leqslant u_i} f_i(x_i)$ are upper bounds on the objective value of any feasible solution of NAKP and NPKP, respectively, which we denote uniformly by $\text{ub}_\otimes$. We then define $u_\otimes := \lfloor n \cdot \log_{1+\varepsilon}(\text{ub}_\otimes) \rfloor + 1$ and consider the following $u_\otimes + 1$ intervals, similar to partitions used in other approximation algorithms (e.g., in [EKP02] for the classical knapsack problem):

$$[0, 1), [1, (1+\varepsilon)^{1/n}), [(1+\varepsilon)^{1/n}, (1+\varepsilon)^{2/n}), \dots, [(1+\varepsilon)^{(u_\otimes - 1)/n}, (1+\varepsilon)^{u_\otimes/n}) \tag{6.6}$$

Note that the number of these intervals is polynomial in the input size of the instance and in $1/\varepsilon$ since $u_\otimes \in \mathcal{O}(n \cdot 1/\varepsilon \cdot \log(\text{ub}_\otimes))$, and that the intervals cover the whole range $\{0, \dots, \text{ub}_\otimes\}$ of possible objective values, where the last interval contains $\text{ub}_\otimes$. The lower bounds of the intervals are particularly important in our algorithm, which is why we introduce the set

$$\text{LB}_{\otimes,\varepsilon} := \{0, 1, (1+\varepsilon)^{1/n}, \dots, (1+\varepsilon)^{(u_\otimes - 1)/n}\}.$$

The idea of the algorithm is to dynamically compute solutions and round their objective values down to the nearest value in $\mathrm{LB}_{\otimes,\varepsilon}$. Therefore, for any value $a \geqslant 0$, we denote the largest value $v \in \mathrm{LB}_{\otimes,\varepsilon}$ such that $v \leqslant a$ by $\lfloor a \rfloor_\varepsilon$ in the following.

Our FPTAS is stated in Algorithm 11. During the algorithm, an array $W[\cdot]$ indexed by the values in $\mathrm{LB}_{\otimes,\varepsilon}$ is populated such that, at termination of the algorithm, we have $W[v] = w$ for $v \in \mathrm{LB}_{\otimes,\varepsilon}$ and $w \in \mathbb{Q}$ if and only if there exists a (not necessarily feasible) solution with weight w and objective value *at least* v.

Algorithm 11: FPTAS

```
Procedure FPTAS()
    $W[v] := +\infty$ for all $v \in \mathrm{LB}_{\otimes,\varepsilon}$
    # Process item 1
    for $x_1 = 0, \dots, u_1$ do
        $v := \lfloor f_1(x_1) \rfloor_\varepsilon$
        $W[v] := \min\{W[v], g_1(x_1)\}$
    # Process item $i$
    for $i = 2, \dots, n$ do
        $W'[v] := +\infty$ for all $v \in \mathrm{LB}_{\otimes,\varepsilon}$
        for $x_i = 0, \dots, u_i$ do
            for $v \in \mathrm{LB}_{\otimes,\varepsilon}$ do
                $v' := \lfloor v \otimes f_i(x_i) \rfloor_\varepsilon$
                $W'[v'] := \min\{W'[v'], W[v] + g_i(x_i)\}$
        for $v \in \mathrm{LB}_{\otimes,\varepsilon}$ do
            $W[v] := W'[v]$
    return maximum value $v \in \mathrm{LB}_{\otimes,\varepsilon}$ such that $W[v] \leqslant C$
```

The following proposition shows that Algorithm 11 computes a $(1+\varepsilon)$-approximation for every reachable profit value of NAKP or NPKP:

Proposition 6.2 The following holds at termination of Algorithm 11: For every solution $x \in \mathbb{Z}^n$ with objective value $\bar{v}$ and weight $\bar{w}$, there exists a value $v \in \mathrm{LB}_{\otimes,\varepsilon}$ with $W[v] \leqslant \bar{w}$ and $(1+\varepsilon) \cdot v \geqslant \bar{v}$.

Proof. For $i \in \{1, \dots, n\}$, we refer to a vector $\bar{x} \in \mathbb{Z}^i$ with $0 \leqslant \bar{x}_k \leqslant u_k$ for $k = 1, \dots, i$ as a *partial solution* for items $1, \dots, i$ with objective value $\bigotimes_{k=1}^{i} f_k(\bar{x}_k)$ and weight $\sum_{k=1}^{i} g_k(\bar{x}_k)$. We prove by induction that, after item i has been processed during the algorithm, the following invariant holds:

For each partial solution $\bar{x} \in \mathbb{Z}^i$ for items $1, \dots, i$ with objective value $\bar{v}$ and weight $\bar{w}$, there exists $v \in \mathrm{LB}_{\otimes,\varepsilon}$ such that

$$(1)\ \ W[v] \leqslant \bar{w} \quad \textit{and} \quad (2)\ \ (1+\varepsilon)^{i/n} \cdot v \geqslant \bar{v}.$$

For $i = 1$, we have $\bar{x} = \bar{x}_1 \in \mathbb{Z}^1$ for any partial solution $\bar{x}$. Thus, its objective value and weight are given as $\bar{v} = f_1(\bar{x}_1)$ and $\bar{w} = g_1(\bar{x}_1)$, respectively. Hence, when lines 5 and 6 are executed within the algorithm in the iteration for $x_1 = \bar{x}_1$ of the for loop starting in line 4, we obtain $v = \lfloor f_1(\bar{x}_1) \rfloor_\varepsilon = \lfloor \bar{v} \rfloor_\varepsilon \geqslant (1+\varepsilon)^{-1/n} \cdot \bar{v}$ and $W[v] \leqslant g_1(\bar{x}_1) = \bar{w}$, which proves that the invariant holds for $i = 1$.

Now consider the situation after item $i \geqslant 2$ has been processed during the algorithm and assume that the invariant holds for $i - 1$. Let $\bar{x} = (\bar{x}_1, \dots, \bar{x}_{i-1}, \bar{x}_i)$ be a partial solution for items $1, \dots, i$ with objective value $\bar{v}$ and weight $\bar{w}$. From this partial solution, we derive the partial solution $\bar{x}^- = (\bar{x}_1, \dots, \bar{x}_{i-1})$ for items $1, \dots, i-1$ with objective value $\bar{v}^-$ and weight $\bar{w}^-$. By definition of $\bar{x}^-$, we then have $\bar{v} = \bar{v}^- \otimes f_i(\bar{x}_i)$ and $\bar{w} = \bar{w}^- + g_i(\bar{x}_i)$. Moreover, since the invariant holds for $i-1$, we know that after item $i-1$ has been processed, there exists $v^- \in \mathrm{LB}_{\otimes,\varepsilon}$ such that

$$(1)\;\; W[v^-] \leqslant \bar{w}^- \quad \text{ and } \quad (2)\;\; (1+\varepsilon)^{(i-1)/n} \cdot v^- \geqslant \bar{v}^-.$$

Hence, when lines 12 and 13 are executed during the processing of item i for $x_i = \bar{x}_i$ and $v = v^-$, we obtain

$$\begin{aligned} v' = \left\lfloor v^- \otimes f_i(\bar{x}_i) \right\rfloor_\varepsilon &\geqslant \left\lfloor \frac{1}{(1+\varepsilon)^{(i-1)/n}} \cdot \bar{v}^- \otimes f_i(\bar{x}_i) \right\rfloor_\varepsilon \\ &\geqslant \frac{1}{(1+\varepsilon)^{i/n}} \cdot \left(\bar{v}^- \otimes f_i(\bar{x}_i)\right) = \frac{1}{(1+\varepsilon)^{i/n}} \cdot \bar{v} \end{aligned}$$

and

$$W'[v'] \leqslant W[v^-] + g_i(\bar{x}_i) \leqslant \bar{w}^- + g_i(\bar{x}_i) = \bar{w}.$$

Thus, after the copying step in lines 14–15 has been executed, the invariant holds for i.

In order to conclude the proof, note that partial solutions for items $1, \dots, n$ are simply solutions $x \in \mathbb{Z}^n$. Thus, the invariant for $i = n$ implies that, after all items $i = 1, \dots, n$ have been processed during the algorithm, for each solution $x \in \mathbb{Z}^n$ with objective value $\bar{v}$ and weight $\bar{w}$, there exists $v \in \mathrm{LB}_{\otimes,\varepsilon}$ with $W[v] \leqslant \bar{w}$ and $(1+\varepsilon) \cdot v \geqslant \bar{v}$ as claimed. □

Denoting the maximum total number of available item copies by $N := \sum_{i=1}^{n} u_i$, it is easy to see that the running time of Algorithm 11 is in

$$\mathcal{O}\left(N \cdot u_\otimes\right) = \mathcal{O}\left(N \cdot n \cdot \frac{1}{\varepsilon} \cdot \log\left(\mathrm{ub}_\otimes\right)\right).$$

Using that $\log(\mathrm{ub}_+) \in \mathcal{O}(\log(n) + \log(f_{\max}))$ and $\log(\mathrm{ub}_\cdot) \in \mathcal{O}(n \cdot \log(f_{\max}))$, we obtain a running time in $\mathcal{O}\left(N \cdot n \cdot 1/\varepsilon \cdot (\log(n) + \log(f_{\max}))\right)$ for NAKP and $\mathcal{O}\left(N \cdot n^2 \cdot 1/\varepsilon \cdot \log(f_{\max})\right)$ for NPKP, where $f_{\max} := \max\limits_{i \in \mathcal{N}} \max\limits_{0 \leqslant x_i \leqslant u_i} f_i(x_i)$ denotes the largest value of any profit function. Together with Proposition 6.2, this yields the following theorem:

Theorem 6.3 For NAKP and NPKP, there exists an FPTAS with running time in $\mathcal{O}(N \cdot n \cdot 1/\varepsilon \cdot (\log(n) + \log(f_{\max})))$ and $\mathcal{O}\left(N \cdot n^2 \cdot 1/\varepsilon \cdot \log(f_{\max})\right)$, respectively, where $f_{\max} := \max\limits_{i \in \mathcal{N}} \max\limits_{0 \leqslant x_i \leqslant u_i} f_i(x_i)$.

6.4 Extension to Multiple Objectives

In this section, the dynamic programming algorithm from Section 6.3 is extended to the multi-objective nonlinear product-sum knapsack problem (MNPSKP). The extension follows the approach pursued, e.g., in [EKP02], but allows for arbitrary combinations of product and sum objective functions. To this end, for each objective function f^j, $j = 1, \dots, p$, we define an upper bound $\mathrm{ub}_{\otimes}^j$ as in the single-objective case and cover the range $\{0, \dots, \mathrm{ub}_{\otimes}^j\}$ of possible values of f_j by $u_{\otimes}^j + 1$ intervals of geometrically increasing width as in (6.6), where $u_{\otimes}^j := \lfloor n \cdot \log_{1+\varepsilon} \mathrm{ub}_{\otimes}^j \rfloor + 1$. The resulting set of lower bounds of these intervals for the j-th objective is denoted by $\mathrm{LB}_{\otimes,\varepsilon}^{(j)}$.

In our MFPTAS stated in Algorithm 12, the dynamic programming array $W[\cdot]$ introduced in Section 6.3 is extended to a p-dimensional array indexed by vectors $(v_1, \dots, v_p) \in \mathrm{LB}_{\otimes,\varepsilon}^{(1)} \times \dots \times \mathrm{LB}_{\otimes,\varepsilon}^{(p)} =: \mathcal{LB}_{\otimes,\varepsilon}$ such that $W[v_1, \dots, v_p] = w$ for some $w \in \mathbb{Q}$ at termination of the algorithm if and only if there exists a (not necessarily feasible) solution with weight w and j-th objective value *at least* v_j for $j = 1, \dots, p$.

Analogous to Algorithm 11, the array is initialized when processing item 1 by considering all possible numbers $x_1 = 0, \dots, u_1$ of copies of item 1 that could be packed into the knapsack. The update operation performed when processing each of the items $2, \dots, n$ is also generalized in the natural way. Here, the for loops starting in lines 12 and 16 now iterate over all p-tuples $(v_1, \dots, v_p)$ in $\mathcal{LB}_{\otimes,\varepsilon}$. Moreover, a rounded-down updated value v_j' now needs to be computed for each $j = 1, \dots, p$, and the resulting p-tuple $(v_1', \dots, v_p')$ is then used as a vector of array indices in line 15. Additionally, the filtering step in lines 18–20 that filters out array entries corresponding to infeasible solutions replaces the selection of a feasible solution with maximum objective value at the end of the algorithm.

The following proposition, whose proof extends the proof of Proposition 6.2, establishes that Algorithm 12 computes a $(1 + \varepsilon)$-approximation for every reachable vector of profit values:

Proposition 6.4 The following holds at termination of Algorithm 12: For every feasible solution $x \in \mathbb{Z}^n$ with objective values $\bar{v}_1, \dots, \bar{v}_p$ and weight $\bar{w}$, there exists a p-tuple $(v_1, \dots, v_p)$ in $\mathcal{LB}_{\otimes,\varepsilon}$ with $W[v_1, \dots, v_p] \leqslant \bar{w}$ and $(1 + \varepsilon) \cdot v_j \geqslant \bar{v}_j$ for $j = 1, \dots, p$.

Proof. The proof extends the proof of Proposition 6.2 to the case of multiple objectives. Consequently, for $i \in \{1, \dots, n\}$, each partial solution $\bar{x} \in \mathbb{Z}^i$ now has p objective values $\bigotimes_{k=1}^{i} f_k^j(\bar{x}_k)$ for $j = 1, \dots, p$ and we prove by induction that the following invariant holds after item i has been processed during the algorithm:

For each partial solution $\bar{x} \in \mathbb{Z}^i$ for items $1, \dots, i$ with objective values $\bar{v}_1, \dots, \bar{v}_p$

Algorithm 12: MFPTAS

```
1  Procedure MFPTAS()
2      $W[v_1,\dots,v_p] := +\infty$ for all $(v_1,\dots,v_p) \in \mathcal{LB}_{\otimes,\varepsilon}$
3      # Process item 1
4      for $x_1 = 0,\dots,u_1$ do
5          for $j = 1,\dots,p$ do
6              $v_j := \left\lfloor f_1^j(x_1) \right\rfloor_\varepsilon$
7          $W[v_1,\dots,v_p] := \min\{W[v_1,\dots,v_p], g_1(x_1)\}$
8      # Process item $i$
9      for $i = 2,\dots,n$ do
10         $W'[v_1,\dots,v_p] := +\infty$ for all $(v_1,\dots,v_p) \in \mathcal{LB}_{\otimes,\varepsilon}$
11         for $x_i = 0,\dots,u_i$ do
12             for $(v_1,\dots,v_p) \in \mathcal{LB}_{\otimes,\varepsilon}$ do
13                 for $j = 1,\dots,p$ do
14                     $v'_j := \left\lfloor v_j \otimes f_i^j(x_i) \right\rfloor_\varepsilon$
15                 $W'[v'_1,\dots,v'_p] := \min\left\{W'[v'_1,\dots,v'_p], W[v_1,\dots,v_p] + g_i(x_i)\right\}$
16         for $(v_1,\dots,v_p) \in \mathcal{LB}_{\otimes,\varepsilon}$ do
17             $W[v_1,\dots,v_p] := W'[v_1,\dots,v_p]$
18     for $(v_1,\dots,v_p) \in \mathcal{LB}_{\otimes,\varepsilon}$ do
19         if $W[v_1,\dots,v_p] > C$ then
20             $W[v_1,\dots,v_p] := +\infty$
```

and weight $\bar{w}$, there exists a p-tuple $(v_1,\dots,v_p)$ in $\mathcal{LB}_{\otimes,\varepsilon}$ such that

$$(1)\;\; W[v_1,\dots,v_p] \leqslant \bar{w} \quad \textit{and} \quad (2)\;\; (1+\varepsilon)^{i/n} \cdot v_j \geqslant \bar{v}_j \textit{ for } j = 1,\dots,p.$$

For $i = 1$, we have $\bar{x} = \bar{x}_1 \in \mathbb{Z}^1$ for any partial solution $\bar{x}$. Thus, for $j = 1,\dots,p$, its j-th objective value is given as $\bar{v}_j = f_1^j(\bar{x}_1)$ and its weight is given as $\bar{w} = g_1(\bar{x}_1)$. Hence, when lines 6 and 7 are executed within the algorithm in the iteration for $x_1 = \bar{x}_1$ of the for loop starting in line 4, we obtain $v_j = \lfloor f_1^j(\bar{x}_1)\rfloor_\varepsilon = \lfloor \bar{v}_j \rfloor_\varepsilon \geqslant (1+\varepsilon)^{-1/n} \cdot \bar{v}_j$ for $j = 1,\dots,p$ and $W[v_1,\dots,v_p] \leqslant g_1(\bar{x}_1) = \bar{w}$, which proves that the invariant holds for $i = 1$.

Now consider the situation after item $i \geqslant 2$ has been processed during the algorithm and assume that the invariant holds for $i-1$. Let $\bar{x} = (\bar{x}_1,\dots,\bar{x}_{i-1},\bar{x}_i)$ be a partial solution for items $1,\dots,i$ with objective values $\bar{v}_1,\dots,\bar{v}_p$ and weight $\bar{w}$. From this partial solution, we derive the partial solution $\bar{x}^- = (\bar{x}_1,\dots,\bar{x}_{i-1})$ for items $1,\dots,i-1$ with objective values $\bar{v}_1^-,\dots,\bar{v}_p^-$ and weight $\bar{w}^-$. By definition of $\bar{x}^-$, we then have $\bar{v}_j = \bar{v}_j^- \otimes f_i^j(\bar{x}_i)$ for $j = 1,\dots,p$ and $\bar{w} = \bar{w}^- + g_i(\bar{x}_i)$. Moreover, since the invariant holds for $i-1$, we know that after item $i-1$ has been processed, there exists $(v_1^-,\dots,v_p^-) \in \mathcal{LB}_{\otimes,\varepsilon}$ such that

$$(1)\;\; W[v_1^-,\dots,v_p^-] \leqslant \bar{w}^- \quad \text{and} \quad (2)\;\; (1+\varepsilon)^{i-1/n} \cdot v_j^- \geqslant \bar{v}_j^- \text{ for } j = 1,\dots,p.$$

Hence, when lines 14 and 15 are executed during the processing of item i for $x_i = \bar{x}_i$ and $(v_1, \ldots, v_p) = (v_1^-, \ldots, v_p^-)$, we obtain

$$\begin{aligned} v_j' &= \left\lfloor v_j^- \otimes f_i^j(\bar{x}_i) \right\rfloor_\varepsilon \geqslant \left\lfloor \frac{1}{(1+\varepsilon)^{i-1/n}} \cdot \bar{v}_j^- \otimes f_i^j(\bar{x}_i) \right\rfloor_\varepsilon \\ &\geqslant \frac{1}{(1+\varepsilon)^{i/n}} \cdot \left(\bar{v}_j^- \otimes f_i^j(\bar{x}_i) \right) = \frac{1}{(1+\varepsilon)^{i/n}} \cdot \bar{v}_j \text{ for } j = 1, \ldots, p \end{aligned}$$

and

$$W'[v_1', \ldots, v_p'] \leqslant W[v_1^-, \ldots, v_p^-] + g_i(\bar{x}_i) \leqslant \bar{w}^- + g_i(\bar{x}_i) = \bar{w}.$$

Thus, after the copying step in lines 16–17 has been executed, the invariant holds for i.

Since partial solutions for items $1, \ldots, n$ are simply solutions $x \in \mathbb{Z}^n$, the invariant for $i = n$ shows the claim (the filtering step in lines 18–20 only filters out array entries corresponding to infeasible solutions). □

By Proposition 6.4, the set of feasible solutions corresponding to the array entries at termination of Algorithm 12 is a $(1+\varepsilon)$-approximate Pareto set. Moreover, similar to the running time analysis of Algorithm 11, it is easy to see that the running time of Algorithm 12 is in

$$\begin{aligned} &\mathcal{O}\left(N \cdot \prod_{j=1}^{p} u_\otimes^j \right) \\ &= \mathcal{O}\left(N \cdot \left(\frac{n}{\varepsilon}\right)^p \cdot \prod_{j=1}^{p} \log\left(\mathrm{ub}_\otimes^j\right) \right) \\ &= \mathcal{O}\left(N \cdot \left(\frac{n}{\varepsilon}\right)^p \cdot \prod_{j=1}^{p'} \log(f_{\max}^j) \right) \\ &= \mathcal{O}\left(N \cdot \left(\frac{n}{\varepsilon}\right)^p \cdot \prod_{j=1}^{p'} \log\left(\mathrm{ub}_\cdot^j\right) \cdot \prod_{j=p'+1}^{p} \log\left(\mathrm{ub}_+^j\right) \right) \\ &= \mathcal{O}\left(N \cdot \left(\frac{n}{\varepsilon}\right)^p \cdot n^{p'} \cdot \prod_{j=1}^{p'} \log\left(f_{\max}^j\right) \cdot \prod_{j=p'+1}^{p} \left(\log\left(f_{\max}^j\right) + \log(n)\right) \right) \end{aligned}$$

where, for $j = 1, \ldots, p$, the term $f_{\max}^j := \max_{i \in \mathcal{N}} \max_{0 \leqslant x_i \leqslant u_i} f_i^j(x_i)$ denotes the largest value of any item's j-th profit function. Note that, since the number p of objective functions is a constant (which is a standard assumption in multi-objective optimization), this running time bound is still polynomial in the encoding length of the input (and in $1/\varepsilon$). Together with Proposition 6.4, this yields the following theorem:

Theorem 6.5 There exists an MFPTAS for MNPSKP with running time in

$$\mathcal{O}\left(N \cdot \left(\frac{n}{\varepsilon}\right)^p \cdot n^{p'} \cdot \prod_{j=1}^{p'} \log\left(f^j_{\max}\right) \cdot \prod_{j=p'+1}^{p} \left(\log\left(f^j_{\max}\right) + \log(n)\right)\right),$$

where $f^j_{\max} := \max_{i \in \mathcal{N}} \max_{0 \leqslant x_i \leqslant u_i} f^j_i(x_i)$ for $j = 1, \ldots, p$.

In the special case of pure additive or multiplicative objective functions in MNPSKP, i.e., the special case that $p' = 0$ and $p' = p$, respectively, the running time of the algorithm is in $\mathcal{O}\left(N \cdot \left(\frac{n}{\varepsilon}\right)^p \cdot \prod_{j=1}^{p} \left(\log\left(f^j_{\max}\right) + \log(n)\right)\right)$ and $\mathcal{O}\left(N \cdot \left(\frac{n^2}{\varepsilon}\right)^p \cdot \prod_{j=1}^{p} \log\left(f^j_{\max}\right)\right)$, respectively.

6.5 Applications

In this section, we use the general approximation scheme presented in the previous section to obtain an FPTAS for the 0-1 time-bomb knapsack problem mentioned in the introduction, where a sum and a product are multiplied in the objective function (Section 6.5.1). Moreover, we modify our algorithm so that it can also be applied for the minimization case. The modified algorithm is then used to formally state an FPTAS for the minimization knapsack problem in Section 6.5.2, which cannot be found in the literature in an explicit form so far. As a third application, we consider scenario-based robust optimization problems in Section 6.5.4 and derive an FPTAS for the most general form of a max-min knapsack problem as well as other objectives.

6.5.1 An FPTAS for the 0-1 Time-Bomb Knapsack Problem

The 0-1 *time-bomb knapsack problem* (TBKP) has recently been introduced in [MPS22]. In this extension of the standard 0-1 knapsack problem, each item $i \in \mathcal{N}$ has a nonnegative, integer weight $w_i \in \mathbb{Z}_{\geqslant 0}$, a nonnegative, integer profit $p_i \in \mathbb{Z}_{\geqslant 0}$, and an additional *explosion probability* $q_i \in \mathbb{Q} \cap [0, 1)$. Each item i that is packed into the knapsack explodes with probability q_i, which then destroys the entire content of the knapsack and, thus, reduces the obtained profit to zero. Here, the random (indicator) variable determined by whether an item explodes or not is assumed to be independent of the corresponding variables of all other items. The task in TBKP consists of maximizing the expected profit obtained from the packed items and can be formulated as follows (see [MPS22]):

$$\max \left(\sum_{i=1}^{n} p_i x_i\right) \cdot \left(\prod_{i=1}^{n} (1 - q_i x_i)\right) \tag{6.7}$$

$$\text{s.t.} \sum_{i=1}^{n} w_i x_i \leqslant C \tag{6.8}$$

$$x_i \in \{0, 1\} \qquad \forall i \in \{1, \ldots, n\} \tag{6.9}$$

Obviously, every feasible solution is associated with a certain total profit and a combined explosion probability. The product of these two values constitutes the expected profit of the solution. Therefore, we can also look at the problem in a bi-objective setting, where the first objective function f^1 is related to the profit and the second objective function f^2 to the explosion probability.

To define the resulting bi-objective nonlinear product-sum knapsack problem formally using the notation introduced in Section 6.2.2, we set $p := 2$ and $p' := 1$, $f_i^1(0) := 0$ and $f_i^1(1) := p_i$, $f_i^2(0) := 1$ and $f_i^2(1) := \pi_i := 1 - q_i$, $g_i(0) := 0$ and $g_i(1) := w_i$ as well as $u_i := 1$ for every $i \in \mathcal{N}$ (here, π_i corresponds to the probability that item i does *not* explode). Afterwards, in order to satisfy our assumption of integer-valued profit functions, even though the values $f_i^2(1) = \pi_i$, $i \in \mathcal{N}$, are fractional, we then multiply all values $f_i^2(x_i)$ for $x_i \in \{0, 1\}$ and $i \in \mathcal{N}$ by their lowest common denominator $d \in \mathbb{N}_{>0}$. This yields an equivalent instance since the second objective value of any solution is multiplied by the same positive factor d^n of polynomial encoding length (recall the discussion in Section 6.2.1). The set of feasible solutions of TBKP is then identical to the set of feasible solutions of the resulting *time-bomb bi-objective nonlinear product-sum knapsack problem* (TB-BNPSKP), and a solution's objective function value in TBKP is obtained as $1/d^n$ times the product of its two objective function values in TB-BNPSKP.

In order to obtain an FPTAS for TBKP, we proceed as follows: Given an instance of TBKP and $\varepsilon > 0$, we apply Algorithm 12 with an error bound of $\varepsilon' := \sqrt{1+\varepsilon} - 1$ to the corresponding instance of TB-BNPSKP. If we let x^* denote an optimal solution of the TBKP instance, then the $(1+\varepsilon')$-approximate Pareto set obtained from Algorithm 12 must contain a feasible solution x^A that $(1+\varepsilon')$-approximates x^*, i.e., such that $(1+\varepsilon') \sum_{i=1}^n f_i^1(x_i^A) \geqslant \sum_{i=1}^n f_i^1(x_i^*)$ and $(1+\varepsilon') \prod_{i=1}^n f_i^2(x_i^A) \geqslant \prod_{i=1}^n f_i^2(x_i^*)$. Consequently, since $(1+\varepsilon')^2 = 1+\varepsilon$, the objective value of x^A in the TBKP instance satisfies:

$$\begin{aligned}
&(1+\varepsilon) \cdot \left(\sum_{i=1}^n p_i x_i^A \right) \cdot \left(\prod_{i=1}^n (1 - q_i x_i^A) \right) \\
&= (1+\varepsilon') \cdot \left(\sum_{i=1}^n f_i^1(x_i^A) \right) \cdot (1+\varepsilon') \cdot \frac{1}{d^n} \cdot \left(\prod_{i=1}^n f_i^2(x_i^A) \right) \\
&\geqslant \left(\sum_{i=1}^n f_i^1(x_i^*) \right) \cdot \frac{1}{d^n} \cdot \left(\prod_{i=1}^n f_i^2(x_i^*) \right) \\
&= \left(\sum_{i=1}^n p_i x_i^* \right) \cdot \left(\prod_{i=1}^n (1 - q_i x_i^*) \right)
\end{aligned}$$

Thus, the optimal solution x^* is $(1+\varepsilon)$-approximated by x^A in the original TBKP instance. Consequently, selecting the solution with the highest objective value in (6.7) from the $(1+\varepsilon')$-approximate Pareto set obtained from Algorithm 12 yields a $(1+\varepsilon)$-approximate solution for the TBKP instance. Since the running time of this procedure is dominated by the (polynomial) running time of Algorithm 12 applied to the TB-BNPSKP instance and we have $1/\varepsilon' \in \mathcal{O}(1/\varepsilon)$, using Theorem 6.5, this shows:

Theorem 6.6 There exists an FPTAS for the 0-1 time-bomb knapsack problem (TBKP) with running time in $\mathcal{O}\left(n^4 \cdot \left(\frac{1}{\varepsilon}\right)^2 \cdot \log(d) \cdot (\log(p_{\max}) + \log(n))\right)$.

6.5.2 Minimization Knapsack Problems

In this section we consider the minimization counterparts of NAKP and NPKP with the classical minimization knapsack problem as a relevant special case. The problems minNAKP and minNPKP can be derived from their maximization counterparts, by replacing "max" by "min" in (6.1) and replacing (6.2) by

$$\sum_{i=1}^{n} g_i(x_i) \geqslant C, \tag{6.10}$$

whereas (6.3) and (6.4) remain unchanged.

In the literature, a special case of minNAKP is considered in [KN09], where the profit functions f_i consist of linear, not necessarily connected pieces, but all weights are unitary, i.e., $g_i(x_i) = x_i$ for all $i \in \mathcal{N}$. They further present an LP-based heuristic for this special case of minNAKP. The same restriction to unitary weights is considered in [LSH94], where an FPTAS is derived for arbitrary nondecreasing profit functions. A special minimization knapsack problem with a linear objective and an Euclidean norm in the weight constraint arising from electrical power systems is considered in [EKN19]. They provide a PTAS, while our approach provides an FPTAS (after taking the square of the Euclidean norm in the constraint).

In the following, we briefly describe how to modify Algorithm 11 in order to derive an FPTAS for minNAKP and minNPKP.

Following the exposition of Section 6.3, we use the same set $\mathrm{LB}_{\otimes,\varepsilon}$ of interval boundaries, but extend it by the value $(1+\varepsilon)^{u_{\otimes}/n}$. In the algorithm, we round objective values $a \geqslant 0$ *up* (instead of down) to the nearest value in $\mathrm{LB}_{\otimes,\varepsilon}$ (which we denote by $\lceil a \rceil_{\varepsilon}$). When comparing two weight values for a dynamic programming entry, we take the maximum between the two (instead of the minimum). In the initialization, infeasible entries are set to $W[v] := -\infty$ or $W'[v] := -\infty$ for all $v \in \mathrm{LB}_{\otimes,\varepsilon}$. The value returned at termination of the algorithm is given by the minimum value $v \in \mathrm{LB}_{\otimes,\varepsilon}$ such that $W[v] \geqslant C$.

It should be noted that, in the approximation of minimization problems, instances with optimal objective value 0 deserve special attention since every approximation algorithm has to return a solution with objective value exactly 0 for each such instance. Our dynamic program has no issue with this aspect since every detected objective value $f_1(x_1) = 0$ or $v \otimes f_i(x_i) = 0$ will remain 0 by definition of the rounding procedure. Thus, no error will accrue in these cases.

A full description of the resulting algorithm (Algorithm 13) as well as the proof of the approximation ratio, both of which are similar to the results presented in Section 6.3, are presented in Appendix A.2. We summarize this discussion in the following analogue of Theorem 6.3.

Theorem 6.7 For minNAKP and minNPKP, there exists an FPTAS with running time in $\mathcal{O}(N \cdot n \cdot {}^1\!/\!_\varepsilon \cdot (\log(n) + \log(f_{\max})))$ and $\mathcal{O}\left(N \cdot n^2 \cdot {}^1\!/\!_\varepsilon \cdot \log(f_{\max})\right)$, respectively, where $f_{\max} := \max_{i \in \mathcal{N}} \max_{0 \leqslant x_i \leqslant u_i} f_i(x_i)$.

The minimization version of the standard 0-1 knapsack problem (KP) is well known as the *minimization knapsack problem* (minKP) and is defined as follows:

$$\min \sum_{i=1}^{n} p_i x_i \tag{6.11}$$

$$\text{s.t.} \sum_{i=1}^{n} w_i x_i \geqslant C \tag{6.12}$$

$$x_i \in \{0,1\} \qquad \forall i \in \{1,\dots,n\} \tag{6.13}$$

Here, p_i and w_i are the integer profit and weight of an item $i \in \mathcal{N}$, respectively. Clearly, minKP is a special case of minNAKP.

Heuristics and approximation algorithms for minKP are presented in [Csi+91; GL79; GJ00]. Surprisingly, an explicit description of an FTPAS for minKP seems to be missing in the literature so far. In the monograph [KPP04, ch.13.3.3], only a vague hint at an FPTAS is given. The existence of an FPTAS for minKP follows from the general framework given in [Woe00]. It can also be deduced from [Kov96] by plugging in an approximation algorithm for minKP with constant approximation ratio as given in [Csi+91]. Note that [GL79] also contains pointers to approximation algorithms for minKP in the early Russian literature. In order to fill the gap of an explicit reference to an FPTAS for minKP, we state the following consequence of Theorem 6.7.

Corollary 6.8 Algorithm 13 gives an FPTAS for minKP with running time in

$$\mathcal{O}\left(n^2 \cdot \frac{1}{\varepsilon} \cdot (\log(n) + \log(p_{\max}))\right).$$

The FPTAS is obtained from Algorithm 13 in Appendix A.2 with $\otimes := +$ by setting $u_i := 1$, $f_i(0) := 0$, $f_i(1) := p_i$, $g_i(0) := 0$, and $g_i(1) := w_i$ for all $i \in \mathcal{N}$.

6.5.3 Extension of the Minimization Version to Multiple Objectives

Analogous to Section 6.2.2, we can define the multi-objective extensions of minNAKP and minNPKP. The multi-objective nonlinear product-sum minimization knapsack problem (min-MNPSKP) is similar to MNPSKP, but the entries of the vector (6.5) of objective values are to be minimized.

Similarly to Section 6.4, we can generalize the FPTAS provided in Algorithm 13 to the multi-objective case by extending the dynamic programming array from one to p dimensions for vectors $(v_1,\dots,v_p) \in \mathcal{LB}_{\otimes,\varepsilon}$. At termination of the corresponding algorithm, an entry $W[v_1,\dots,v_p] = w$ for some $w \in \mathbb{Q}$ represents a (not necessarily feasible) solution with weight w and j-th objective value *at most* v_j for $j = 1,\dots,p$. The resulting algorithm can be derived from Algorithm 12 in a similar way as Algorithm 13 has been derived from Algorithm 11. Without going into further details, we state the following complement of Theorem 6.5.

Corollary 6.9 There exists an MFPTAS for minMNPSKP with the same running time as stated in Theorem 6.5.

Looking at the FPTASs for the single-objective maximization and minimization problems and their extensions to the multi-objective case, one can notice that, without further difficulties, one can also combine maximization and minimization objectives in a multi-objective setting. Clearly, this requires some effort for formally defining dominance and efficient solutions for such a combination of different directions of optimization, but the approximation ratios obtained for each objective can be combined similarly to the cases with uniform directions. The knapsack constraint then makes sense for either direction of the inequality.

6.5.4 Max-Min Versions of the Nonlinear Knapsack Problem

In practical applications, the profit functions in a knapsack problem are usually not known exactly, but can only be estimated. Consequently, it is a natural question whether robust solutions can be found that still perform well when profits are uncertain.

In the max-min version of a nonlinear knapsack problem, we are given a set $\mathcal{S}$ of *scenarios*. For each scenario $S \in \mathcal{S}$, the profit of item $i \in \mathcal{N}$ under scenario S is given by a function $f_i^S : \{0, 1, \dots, u_i\} \to \mathbb{Q}_{\geqslant 0}$, which has the properties stated for the profit functions in Assumption 6.1. The aim of the problem is to find a solution that maximizes the minimum of the profits over all the scenarios, i.e., a solution that performs best-possible in the worst-case scenario.

In the well-known *max-min nonlinear additive knapsack problem* (max-min-NAKP) and in the *max-min nonlinear product knapsack problem* (max-min-NPKP), the objective is given by

$$\max_{x \in \mathbb{Z}^n} \min_{S \in \mathcal{S}} \quad \sum_{i=1}^{n} f_i^S(x_i) \quad \text{or} \quad \prod_{i=1}^{n} f_i^S(x_i),$$

while the constraints are given by (6.2)–(6.4) as in NAKP and NPKP.

It is shown in [ABV09] that any MFPTAS for the multi-objective knapsack problem gives rise to an FPTAS for the max-min-NAKP since there always exists a max-min optimal solution that is efficient. It is straightforward to see that, if the number of scenarios is constant (which is also assumed in [ABV09]), the same reasoning applies also for the usage of an MFPTAS for MNPSKP. Using Algorithm 12 presented in Section 6.4, this yields an FPTAS for the max-min-NAKP and the max-min-NPKP and, thus, extends the previously-mentioned result to objective functions that employ a product instead of a sum objective function.

It is worth noting that – since our FPTAS for MNPSKP can handle arbitrary combinations of sum and product objective functions – the same procedure can also solve a mixed version of the max-min knapsack problem where the operator (sum or product) used in the objective function also depends on the scenario. Further, the procedure can be slightly adapted such that it provides an FPTAS for the max-max version of NPKP or NAKP, in which the aim is to find the solution that performs best in the *best-case* scenario. Moreover, by the same arguments, the minimization procedure presented in Section 6.5.3 yields an FPTAS for the min-max version and for the min-min version of NPKP and NAKP. The min-max version of NAKP is of particular interest since it is a robust version of the well-known minKP, which has been attended in Section 6.5.2.

6.6 Conclusion

In this chapter, a very general version of the knapsack problem with separable, nonlinear profit and weight functions as well as sums and/or products as operators in the objective function is presented. Further, the problem is addressed in its maximization and minimization version in both the single-objective and the multi-objective setting. We present an FPTAS and an MFPTAS for the single-objective and the multi-objective case, respectively. These algorithms are applicable to a wide variety of knapsack problems with separable profit and weight functions studied in the literature. We note that separability of profit contributions can be seen as a natural limit of approximation algorithms since the quadratic knapsack problem (QKP), whose objective function is *not* separable since the pairwise inclusion of two items i and j contributes a profit p_{ij}, does not permit an FPTAS unless $\mathcal{P} = \mathcal{NP}$, and no constant-factor approximation algorithm is known for QKP so far [PS16; Tay16].

Our general results are applicable to a wide variety of specific problems from the literature. For instance, we obtain the first FPTAS for the recently-introduced 0-1 time-bomb knapsack problem. Moreover, our MFPTAS gives rise to FPTASs for a class of knapsack problems occurring in robust optimization, where uncertainty is modelled by a set of scenarios representing possible realizations of input data. Here, the aim in the max-min problem, for example, consists of maximizing the profit guaranteed in every scenario, i.e., even in the worst possible scenario. We present an FPTAS for all four possible combinations, i.e., for the max-min, max-max, min-max, and min-min version of the problem. Further applications may well arise due to the general nature of our setting.

7 | Conclusion

In this thesis, we investigate the problem of choosing best-possible combinations of precautionary measures for pluvial flash floods as well as related interdiction problems on graphs and variations of the knapsack problem. The former problem is studied from a more practical viewpoint, presenting a mixed-integer programming formulation of the problem, a variety of additional methods to reduce the running time making the overall algorithm applicable to practical problems, and the emerging software, which has meanwhile been used by over 30 institutions. The latter problems are examined rather from a theoretical point of view, assessing their complexity and presenting polynomial-time algorithms, approximation algorithms, FPTASs, or MFPTASs.

Thus, after basic concepts about complexity theory, graphs and networks, multicriteria optimization, and approximation algorithms have been introduced in Chapter 2, we present the results of the project AKUT – an acronym for the German translation of "Incentive Systems for Municipal Flood Prevention" – in Chapter 3, where the aim is to compute a combination of precautionary measures that effectively protects the buildings while adhering to a budget constraint and taking the cooperation of local residents into account. The presentation is started with a formal definition of the problem and a description of the input data, which is chosen such that it is available to German municipalities. It is then continued by presenting a combinatorial algorithm computing the water levels that are to be expected if a given combination of precautionary measures is taken. Next, a mixed-integer programming formulation and several presolve techniques are presented. Moreover, it is shown that the mixed-integer programming formulation is indeed a valid formulation of the presented problem and the chapter is concluded with computational results that point out the most important drivers for the quality of the obtained solution and the running time.

In Chapter 4, the network flow interdiction problem, where arcs are to be removed from a network subject to a budget constraint such that the value of a maximum s-t-flow is minimized, is studied. This problem is in a natural way connected to the task of protecting certain nodes in a graph from inflow and, hence, closely related to the problem statement in the project AKUT. We present a $(B+1)$-approximation algorithm for the special case of the network flow interdiction problem, where arcs have unit removal cost and where B is the budget, i.e., the maximum number of arcs that can be removed from the network. To the best of our knowledge, this is the first approximation algorithm for any version of the problem whose approximation ratio does not depend on the size of the network. Further, it is worth noting that, on simple graphs, this approximation ratio dominates the previously best known approximation ratio of $(n-1)$, where n is the number of nodes in the network.

Our work is continued by investigating the temporal shortest path interdiction problem in Chapter 5. Temporal graphs, in which arcs are only available at certain points in time, have lately attracted the interest of the research community since incorporating the time in graph problems is in many cases crucial to obtain realistic models for real-world problems such as the spread of the virus during the COVID-19 pandemic. However, interdiction problems have barely been investigated on temporal graphs so far. We start by presenting four different notions of the term "shortest" in the temporal case and investigate the complexities of the four arising interdiction problems. Interestingly, although the shortest path interdiction problem on static graphs is known to be $\mathcal{NP}$-hard, two of the four versions on temporal graphs are polynomial-time solvable while the other two are $\mathcal{NP}$-hard. However, we show that, on extension-parallel temporal graphs, the latter two versions are also polynomial-time solvable. We conclude the chapter by presenting three extensions of the problem and show how the complexities change under these extensions.

Lastly, we investigate the well-studied non-linear knapsack problem in Chapter 6 and present FPTASs and MFPTASs for highly generalized single- and multi-objective versions of the problem, which cover, with few exceptions, all versions of the (non-linear) knapsack problem studied in the literature so far. To this end, we start with a formal definition of the single- and multi-objective versions and an analysis of our assumptions and their effect on the problem's generality. We continue by presenting the FPTAS and the MFPTAS and their proof of correctness and slightly modify the algorithm such that it also works for the minimization knapsack problem. While vague hints on the existence of an FPTAS or MFPTAS for the minimization knapsack problem have been given in the literature, this is, to the best of our knowledge, the first formal proof and closes an important gap in the literature. We use the obtained MFPTAS to obtain an FPTAS for the recently-introduced 0-1 time-bomb knapsack problem and for some max-min or min-max versions motivated by the field of robust optimization.

Although all chapters are concluded individually, we conclude the work from a more holistic point of view. While adaption to the consequences of climate change is undeniably one of the most central problems of mankind in the 21st century, digital decision support in planning processes of precautionary concepts still often lacks the implementation of combinatorial optimization algorithms and relies on simulation algorithms where only the consequences of one specific scenario can be assessed. We implement a combinatorial optimization algorithm with a corresponding web-application to address the problem of finding good precautionary measures for pluvial flash floods in municipalities and address related problems from a theoretical point of view. The appreciation that is shown from both practical partners and the research community motivates to further address climate-resilience-related problems such as, for example, urban heat development or water management in scenarios of water shortage with methods from combinatorial optimization and other fields of computer science and mathematics. Utilizing nowadays available high-end technology and extending the required theory might be an important piece of the puzzle of adapting to the consequences of climate change – and we are gratified to have contributed to fitting this piece into its correct place.

A Appendix

Appendix A.1 : Constraints of the Mixed-Integer Programming Formulation

We provide the mathematical formulation of the constraints of the MIP presented in Section 3.3.

Water levels at nodes:
Computing excess(v) for each node $v \in V$:

$$\text{excess}(v) = \sum_{r \in \delta^+_{G^{ex}}(v)} f(r) - \sum_{r \in \delta^-_{G^{ex}}(v)} f(r) + \text{rain} \cdot \text{area}(v) \quad \forall v \in V \tag{1}$$

Computing the water level wl(v) at each node $v \in V$:

$$\text{wl}(v) = \frac{\text{excess}(v)}{\text{area}(v)} \quad \forall v \in V \tag{2}$$

Geodesic heights of nodes:
Setting the variable down(v) for each node $v \in V$:

$$\text{down}(v) \geqslant \text{decBasin}(b) \quad \forall v \in V, b \in \mathcal{B}(v) \tag{3}$$

$$\text{down}(v) \geqslant \text{decDitch}(d) \quad \forall v \in V, d \in \mathcal{D}(v) \tag{4}$$

$$\text{down}(v) \leqslant \sum_{b \in \mathcal{B}(v)} \text{decBasin}(b) + \sum_{d \in \mathcal{D}(v)} \text{decDitch}(d) \quad \forall v \in V \tag{5}$$

Setting the variables hdb(b), hdd(d), and hde(e) for $b \in \mathcal{B}$, $d \in \mathcal{D}$, $e \in \mathcal{E}$:

$$\text{hdb}(b) = \text{depth}(b) \cdot \text{decBasin}(b) \quad \forall b \in \mathcal{B} \tag{6}$$

$$\text{hdd}(d) = \text{depth}(d) \cdot \text{decDitch}(d) \quad \forall d \in \mathcal{D} \tag{7}$$

$$\text{hde}(e) = \text{height}(e) \cdot \text{decEmb}(e) \quad \forall e \in \mathcal{E} \tag{8}$$

Computing the maximum height of an embankment for each node $v \in V$:

$$\text{max\_dec}(v) = max(\{\text{hdb}(b) | b \in \mathcal{B}(v)\} \cup \{\text{hdd}(d) | d \in \mathcal{D}(v)\} \cup \{0\}) \quad \forall v \in V \tag{9}$$

$$\text{max\_inc}(v) = \max(\{\text{hde}(e) | e \in \mathcal{E}(v)\} \cup \{0\}) \quad \forall v \in V \tag{10}$$

Setting the geodesic height variable gh(v) for each node $v \in V$:

$$\text{gh}(v) \geqslant \text{GH}(v) - \text{max\_dec}(v) \quad \forall v \in V \tag{11}$$

$$\text{gh}(v) \leqslant \text{GH}(v) + \text{max\_inc}(v) \quad \forall v \in V \tag{12}$$

$$\text{gh}(v) \leqslant \text{GH}(v) - \text{max\_dec}(v) + (1 - \text{down}(v)) \cdot M(v) \quad \forall v \in V \tag{13}$$

$$\text{gh}(v) \geqslant \text{GH}(v) + \text{max\_inc}(v) - \text{down}(v) \cdot M(v) \quad \forall v \in V \tag{14}$$

Arc directions:
Setting the variable od(r) for each arc $r \in R$:

$$\text{od}(r) = 1 \Rightarrow \text{gh}(\alpha(r)) \geqslant \text{gh}(\omega(r)) \quad \forall r \in R \tag{15}$$

$$\text{od}(r) = 0 \Rightarrow \text{gh}(\alpha(r)) < \text{gh}(\omega(r)) \quad \forall r \in R \tag{16}$$

Full arcs:
Setting the auxiliary variables auxO1F1(r), auxO1F0(r), auxO0F1(r), and auxO0F0(r) for each arc $r \in R$:

$$\text{auxO1F1}(r) \geqslant -1 + \text{od}(r) + \text{full}(r) \quad \forall r \in R \tag{17.1}$$

$$\text{auxO1F1}(r) \leqslant \text{full}(r) \quad \forall r \in R \tag{17.2}$$

$$\text{auxO1F1}(r) \leqslant \text{od}(r) \quad \forall r \in R \tag{17.3}$$

$$\text{auxO1F0}(r) \geqslant \text{od}(r) - \text{full}(r) \quad \forall r \in R \tag{18.1}$$

$$\text{auxO1F0}(r) \leqslant 1 - \text{full}(r) \quad \forall r \in R \tag{18.2}$$

$$\text{auxO1F0}(r) \leqslant \text{od}(r) \quad \forall r \in R \tag{18.3}$$

$$\text{auxO0F1}(r) \geqslant -\text{od}(r) + \text{full}(r) \quad \forall r \in R \tag{19.1}$$

$$\text{auxO0F1}(r) \leqslant \text{full}(r) \quad \forall r \in R \tag{19.2}$$

$$\text{auxO0F1}(r) \leqslant 1 - \text{od}(r) \quad \forall r \in R \tag{19.3}$$

$$\text{auxO0F0}(r) \geqslant 1 - \text{od}(r) - \text{full}(r) \quad \forall r \in R \tag{20.1}$$

$$\text{auxO0F0}(r) \leqslant 1 - \text{full}(r) \quad \forall r \in R \tag{20.2}$$

$$\text{auxO0F0}(r) \leqslant 1 - \text{od}(r) \quad \forall r \in R \tag{20.3}$$

Connecting the variables full(r) to the water levels using the auxiliary variables:

$$\text{auxO1F1}(r) = 1 \Rightarrow \text{wl}(\omega(r)) \geqslant \text{gh}(\alpha(r)) - \text{gh}(\omega(r)) \quad \forall r \in R \tag{21}$$
$$\text{auxO1F0}(r) = 1 \Rightarrow \text{wl}(\omega(r)) < \text{gh}(\alpha(r)) - \text{gh}(\omega(r)) \quad \forall r \in R \tag{22}$$
$$\text{auxO0F1}(r) = 1 \Rightarrow \text{wl}(\alpha(r)) \geqslant \text{gh}(\omega(r)) - \text{gh}(\alpha(r)) \quad \forall r \in R \tag{23}$$
$$\text{auxO0F0}(r) = 1 \Rightarrow \text{wl}(\alpha(r)) < \text{gh}(\omega(r)) - \text{gh}(\alpha(r)) \quad \forall r \in R \tag{24}$$

Flooded nodes:
Setting the variable flooded(v) for each node $v \in V$:

$$\text{flooded}(v) = 0 \Rightarrow \text{wl}(v) = 0 \quad \forall v \in V \tag{25}$$
$$\text{flooded}(v) = 1 \Rightarrow \text{wl}(v) > 0 \quad \forall v \in V \tag{26}$$

Active arcs:
Setting the variable active(r) for each arc $r \in R^{\text{ex}}$:

$$\text{active}(r) + \text{active}\left(\overleftarrow{r}\right) = 1 \quad \forall r \in R \tag{27}$$
$$\text{active}(r) = 0 \Rightarrow f(r) = 0 \quad \forall r \in R^{\text{ex}} \tag{28}$$

Flow on arcs that are not full
Setting the auxiliary variables aux_fd(r) and aux_fd$\left(\overleftarrow{r}\right)$ for each arc $r \in R$:

$$\text{aux\_fd}(r) \geqslant \text{active}(r) - \text{full}(r) \quad \forall r \in R \tag{29.1}$$
$$\text{aux\_fd}(r) \leqslant \text{active}(r) \quad \forall r \in R \tag{29.2}$$
$$\text{aux\_fd}(r) \leqslant 1 - \text{full}(r) \quad \forall r \in R \tag{29.3}$$

$$\text{aux\_fd}\left(\overleftarrow{r}\right) \geqslant \text{active}\left(\overleftarrow{r}\right) - \text{full}(r) \quad \forall r \in R \tag{30.1}$$
$$\text{aux\_fd}\left(\overleftarrow{r}\right) \leqslant \text{active}\left(\overleftarrow{r}\right) \quad \forall r \in R \tag{30.2}$$
$$\text{aux\_fd}\left(\overleftarrow{r}\right) \leqslant 1 - \text{full}(r) \quad \forall r \in R \tag{30.3}$$

Distributing the outflow of each node $v \in V$ among its outgoing arcs in the extended graph that are active and not full:

$$\text{aux\_fd}(r_2) = 1 \Rightarrow f(r_1) \leqslant \frac{\text{ratio}(r_1)}{\text{ratio}(r_2)} \cdot f(r_2) \quad \forall v \in V, r_1, r_2 \in \delta^+_{G^{\text{ex}}}(v) \tag{31.1}$$
$$\text{aux\_fd}(r_1) = 1 \Rightarrow f(r_2) \leqslant \frac{\text{ratio}(r_2)}{\text{ratio}(r_1)} \cdot f(r_1) \quad \forall v \in V, r_1, r_2 \in \delta^+_{G^{\text{ex}}}(v) \tag{31.2}$$

For each arc $r \in R$ that is not full, the water level at the higher of the nodes $\alpha(r)$ and $\omega(r)$

must be zero:

$$\text{auxO1F0}(r) = 1 \Rightarrow \text{wl}(\alpha(r)) = 0 \quad \forall r \in R \tag{32}$$

$$\text{auxO0F0}(r) = 1 \Rightarrow \text{wl}(\omega(r)) = 0 \quad \forall r \in R \tag{33}$$

Water cannot flow on non-full uphill arcs:

$$\text{auxO1F0}(r) = 1 \Rightarrow \text{active}\left(\overleftarrow{r}\right) = 0 \quad \forall r \in R \tag{34}$$

$$\text{auxO0F0}(r) = 1 \Rightarrow \text{active}(r) = 0 \quad \forall r \in R \tag{35}$$

Flow on full arcs:
Setting the flow on each full arc $r \in R$ indirectly by connecting the water levels at its start node and its end node:

$$\text{full}(r) = 1 \Rightarrow \text{gh}(\alpha(r)) + \text{wl}(\alpha(r)) = \text{gh}(\omega(r)) + \text{wl}(\omega(r)) \quad \forall r \in R \tag{36}$$

Maximum water levels at buildings:
Bounding the maximum water level variable max_wl(β) from below for each building $\beta \in B$:

$$\text{max\_wl}(\beta) \geqslant \text{wl}(v) \quad \forall \beta \in B, v \in V(\beta) \tag{37}$$

Hazard classes of buildings:
Setting a hazard class for each building $\beta \in B$ via its maximum water level:

$$\sum_{k=0}^{4} \text{hc}(k, \beta) = 1 \quad \forall \beta \in B \tag{38}$$

$$\text{hc}(k, \beta) = 1 \Rightarrow \text{max\_wl}(\beta) \leqslant \text{thresholdWL}(k) \quad \forall \beta \in B, k \in \{0, 1, 2, 3\} \tag{39}$$

Budget constraint:

$$\sum_{b \in \mathcal{B}} \text{cost}(b) \cdot \text{decBasin}(b) + \sum_{d \in \mathcal{D}} \text{cost}(d) \cdot \text{decDitch}(d) + \sum_{e \in \mathcal{E}} \text{cost}(e) \cdot \text{decEmb}(e) \leqslant \text{budget} \tag{40}$$

Incentives for actors:
Enforcing the given upper bounds on the incentives required for cooperation of actors and ensuring that no actions are taken on properties of actors that do not cooperate at all:

$$\sum_{p \in P_{\text{yellow}}} \text{action}(p) + \sum_{p \in P_{\text{red}}} \text{action}(p) \leqslant \text{maxAllowedYellow} + \text{maxAllowedRed} \tag{41}$$

$$\sum_{p \in P_{\text{red}}} \text{action}(p) \leqslant \text{maxAllowedRed} \tag{42}$$

$$\text{action}(p) = 0 \quad \forall p \in P_{\text{black}} \tag{43}$$

$$\text{decBasin}(b) \leqslant \text{action}(p) \quad \forall b \in \mathcal{B}, p \in P : b \text{ is located on } p \tag{44.1}$$
$$\text{decDitch}(d) \leqslant \text{action}(p) \quad \forall d \in \mathcal{D}, p \in P : d \text{ is located on } p \tag{44.2}$$
$$\text{decEmb}(e) \leqslant \text{action}(p) \quad \forall e \in \mathcal{E}, p \in P : e \text{ is located on } p \tag{44.3}$$

Valid inequalities:
The first arc in a pair of consecutive original-direction (i.e., downhill) arcs can only be full if the second arc is full as well:

$$\text{full}(r_2) \geqslant \text{full}(r_1) - (2 - \text{od}(r_1) - \text{od}(r_2)) \quad \forall r_1, r_2 \in R : \omega(r_1) = \alpha(r_2) \tag{45}$$

If node $v \in V$ is flooded, then each arc $r \in \delta^+_{G^{\text{ex}}}(v)$ with $\text{gh}(v) > \text{gh}(\omega(r))$ must be full:

$$\text{flooded}(v) = 1 \Rightarrow \text{full}(r) \geqslant \text{od}(r) \qquad \forall v \in V, r \in \delta^+_G(v) \tag{46.1}$$
$$\text{flooded}(v) = 1 \Rightarrow \text{full}(r) \geqslant 1 - \text{od}(r) \qquad \forall v \in V, r \in \delta^-_G(v) \tag{46.2}$$

If node $v \in V$ is not flooded, then no arc $r \in \delta^-_{G^{\text{ex}}}(v)$ with $\text{gh}(v) < \text{gh}(\alpha(r))$ can be full:

$$\text{flooded}(v) = 0 \Rightarrow \text{full}(r) \leqslant 1 - \text{od}(r) \qquad \forall v \in V, r \in \delta^-_G(v) \tag{47.1}$$
$$\text{flooded}(v) = 0 \Rightarrow \text{full}(r) \leqslant \text{od}(r) \qquad \forall v \in V, r \in \delta^+_G(v) \tag{47.2}$$

Appendix A.2 : An FPTAS for minNAKP and minNPKP

This section contains a full description of our FPTAS for minNAKP and minNPKP (Algorithm 13) together with the proof of its approximation ratio. The algorithm and the proof are similar to Algorithm 11 and the proof of Proposition 6.2, respectively.

Algorithm 13: FPTAS for Minimization Nonlinear Knapsack

```
1  Procedure FPTASmin()
2      $W[v] := -\infty$ for all $v \in \mathrm{LB}_{\otimes,\varepsilon}$
3      # Process item 1
4      for $x_1 = 0, \dots, u_1$ do
5          $v := \lceil f_1(x_1) \rceil_\varepsilon$
6          $W[v] := \max\{W[v], g_1(x_1)\}$
7      # Process item $i$
8      for $i = 2, \dots, n$ do
9          $W'[v] := -\infty$ for all $v \in \mathrm{LB}_{\otimes,\varepsilon}$
10         for $x_i = 0, \dots, u_i$ do
11             for $v \in \mathrm{LB}_{\otimes,\varepsilon}$ do
12                 $v' := \lceil v \otimes f_i(x_i) \rceil_\varepsilon$
13                 $W'[v'] := \max\{W'[v'], W[v] + g_i(x_i)\}$
14         for $v \in \mathrm{LB}_{\otimes,\varepsilon}$ do
15             $W[v] := W'[v]$
16     return minimum value $v \in \mathrm{LB}_{\otimes,\varepsilon}$ such that $W[v] \geqslant C$
```

To prove the approximation ratio of Algorithm 13, we state the following proposition.

Proposition A.1 The following holds at termination of Algorithm 13: For every solution $x \in \mathbb{Z}^n$ with objective value $\bar{v}$ and weight $\bar{w}$, there exists $v \in \mathrm{LB}_{\otimes,\varepsilon}$ with $W[v] \geqslant \bar{w}$ and $v \leqslant (1+\varepsilon) \cdot \bar{v}$.

Proof. The proof is along the same lines as the proof of Proposition 6.2 with slight modifications. Here, we only state the invariant and prove its correctness, which also works analogously to the proof of Proposition 6.2. To this end, we prove by induction that, after item i has been processed during the algorithm, the following invariant holds:

For each partial solution $\bar{x} \in \mathbb{Z}^i$ for items $1, \dots, i$ with objective value $\bar{v}$ and weight $\bar{w}$, there exists $v \in \mathrm{LB}_{\otimes,\varepsilon}$ such that

$$(1)\ \ W[v] \geqslant \bar{w} \quad \textit{and} \quad (2)\ \ v \leqslant (1+\varepsilon)^{i/n} \cdot \bar{v}.$$

For $i = 1$, we have $\bar{x} = \bar{x}_1 \in \mathbb{Z}^1$ for any partial solution $\bar{x}$. Thus, its objective value and weight are given as $\bar{v} = f_1(\bar{x}_1)$ and $\bar{w} = g_1(\bar{x}_1)$, respectively. Hence, when lines 5 and 6 are executed within the algorithm in the iteration for $x_1 = \bar{x}_1$ of the for loop starting in line 4, we obtain $v = \lceil f_1(\bar{x}_1) \rceil_\varepsilon = \lceil \bar{v} \rceil_\varepsilon \leqslant (1+\varepsilon)^{1/n} \cdot \bar{v}$ and $W[v] \geqslant g_1(\bar{x}_1) = \bar{w}$, which proves that the invariant holds for $i = 1$.

Now consider the situation after item $i \geqslant 2$ has been processed during the algorithm and assume that the invariant holds for $i-1$. Let $\bar{x} = (\bar{x}_1, \dots, \bar{x}_{i-1}, \bar{x}_i)$ be a partial solution for items $1, \dots, i$ with objective value $\bar{v}$ and weight $\bar{w}$. From this partial solution, we derive the partial solution $\bar{x}^- = (\bar{x}_1, \dots, \bar{x}_{i-1})$ for items $1, \dots, i-1$ with objective value $\bar{v}^-$ and weight $\bar{w}^-$. By definition of $\bar{x}^-$, we then have $\bar{v} = \bar{v}^- \otimes f_i(\bar{x}_i)$ and $\bar{w} = \bar{w}^- + g_i(\bar{x}_i)$. Moreover, since the invariant holds for $i-1$, we know that after item $i-1$ has been processed, there exists $v^- \in \mathrm{LB}_{\otimes,\varepsilon}$ such that

$$(1)\ W[v^-] \geqslant \bar{w}^- \quad \text{and} \quad (2)\ v^- \leqslant (1+\varepsilon)^{(i-1)/n} \cdot \bar{v}^-.$$

Hence, when lines 12 and 13 are executed during the processing of item i for $x_i = \bar{x}_i$ and $v = v^-$, we obtain

$$\begin{aligned} v' = \left\lceil v^- \otimes f_i(\bar{x}_i) \right\rceil_\varepsilon &\leqslant \left\lceil (1+\varepsilon)^{(i-1)/n} \cdot \bar{v}^- \otimes f_i(\bar{x}_i) \right\rceil_\varepsilon \\ &\leqslant (1+\varepsilon)^{i/n} \cdot \left(\bar{v}^- \otimes f_i(\bar{x}_i)\right) = (1+\varepsilon)^{i/n} \cdot \bar{v} \end{aligned}$$

and

$$W'[v'] \geqslant W[v^-] + g_i(\bar{x}_i) \geqslant \bar{w}^- + g_i(\bar{x}_i) = \bar{w}.$$

Thus, the after the copying step in lines 14–15 has been executed, the invariant holds for i. □

It is easy see that Theorem 6.7 follows from Proposition A.1. Note that the structure of Algorithm 13 is identical to that of Algorithm 11 and, therefore, the same bound on the running time applies.

B List of Figures

C List of Tables

Bibliography

[AMO93] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network Flows.* Prentice Hall, 1993. DOI: 10.1137/1037020.

[ABV09] H. Aissi, C. Bazgan, and D. Vanderpooten. "Min–max and min–max regret versions of combinatorial optimization problems: A survey". In: *European Journal of Operational Research* 197.2 (2009), 427–438. DOI: 10.1016/j.ejor.2008.09.012.

[ATW11] I. Akgün, B. C. Tansel, and R. K. Wood. "The Multi-Terminal Maximum-Flow Network-Interdiction Problem". In: *European Journal of Operational Research* 211.1 (2011), 241–251. DOI: 10.1016/j.ejor.2010.12.011.

[Akr+20] E. C. Akrida et al. "Temporal vertex cover with a sliding time window". In: *Journal of Computer and System Sciences* 107 (2020), 108–123. DOI: 10.1016/j.jcss.2019.08.002.

[Aus+12] G. Ausiello et al. *Complexity and approximation: Combinatorial optimization problems and their approximability properties.* Springer Science & Business Media, 2012. DOI: 10.1007/978-3-642-58412-1.

[BGV89] M. O. Ball, B. L. Golden, and R. V. Vohra. "Finding the most vital arcs in a network". In: *Operations Research Letters* 8.2 (1989), 73–76. DOI: 10.1016/0167-6377(89)90003-5.

[BKS95] A. Bar-Noy, S. Khuller, and B. Schieber. *The complexity of finding most vital arcs and nodes.* Tech. rep. CS-TR-3539. University of Maryland, 1995.

[BRP21] B. M. Behring, A. Rizzo, and M. Porfiri. "How adherence to public health measures shapes epidemic spreading: A temporal network model". In: *Chaos: An Interdisciplinary Journal of Nonlinear Science* 31.4 (2021), 043115. DOI: 10.1063/5.0041993.

[Ben+20] M. Bentert et al. "Efficient computation of optimal temporal walks under waiting-time constraints". In: *Applied Network Science* 5.1 (2020), 73. DOI: 10.1007/s41109-020-00311-0.

[Ben11] C. Bentz. "On the hardness of finding near-optimal multicuts in directed acyclic graphs". In: *Theoretical Computer Science* 412.39 (2011), 5325–5332. DOI: 10.1016/j.tcs.2011.06.003.

[Ber96] K. A. Berman. "Vulnerability of scheduled networks and a generalization of Menger's Theorem". In: *Networks* 28.3 (1996), 125–134. DOI: 10.1002/(SICI)1097-0037(199610)28:3<125::AID-NET1>3.0.CO;2-P.

[Ble+18] S. Blenkinsop et al. “The INTENSE project: using observations and models to understand the past, present and future of sub-daily rainfall extremes”. In: *Advances in Science and Research* 15 (2018), 117–126. DOI: 10.5194/asr-15-117-2018.

[Blu+98] L. Blum et al. *Complexity and Real Computation.* Springer, 1998. DOI: 10.1007/978-1-4612-0701-6.

[BT20] J. Boeckmann and C. Thielen. “An Approximation Algorithm for Network Flow Interdiction with Unit Costs and Two Capacities”. In: *Proceedings of the 18th Cologne-Twente Workshop on Graphs and Combinatorial Optimization (CTW).* 2020, 157–169. DOI: 10.1007/978-3-030-63072-0_13.

[BT21] J. Boeckmann and C. Thielen. “A (B+1)-approximation for network flow interdiction with unit costs”. In: *Discrete Applied Mathematics* (2021), 1–13. DOI: 10.1016/j.dam.2021.07.008.

[BT23] J. Boeckmann and C. Thielen. *New ways in municipal flood mitigation: A mixed-integer programming approach and its practical application.* Under revision at Operations Research Forum. 2023.

[BTP23] J. Boeckmann, C. Thielen, and U. Pferschy. “Approximating single-and multi-objective nonlinear sum and product knapsack problems”. In: *Discrete Optimization* 48 (2023), 100771. DOI: 10.1016/j.disopt.2023.100771.

[BTW23] J. Boeckmann, C. Thielen, and A. Wittmann. *Complexity of the Temporal Shortest Path Interdiction Problem.* Accepted for the Proceedings of the 2nd Symposium on Algorithmic Foundations of Dynamic Networks (SAND 2023). 2023.

[Bon+15] P. Bonami et al. “On mathematical programming with indicator constraints”. In: *Mathematical Programming* 151 (2015), 191–223. DOI: 10.1007/s10107-015-0891-4.

[Bre+12] R. Brekelmans et al. “Safe dike heights at minimal costs: The nonhomogeneous case”. In: *Operations Research* 60.6 (2012), 1342–1355. DOI: 10.1287/opre.1110.1028.

[BS02] K.M. Bretthauer and B. Shetty. “The nonlinear knapsack problem – algorithms and applications”. In: *European Journal of Operational Research* 138.3 (2002), 459–472. DOI: 10.1016/S0377-2217(01)00179-5.

[BCV21] F. Brunelli, P. Crescenzi, and L. Viennot. “On computing Pareto optimal paths in weighted time-dependent networks”. In: *Information Processing Letters* 168 (2021), 106086. DOI: 10.1016/j.ipl.2020.106086.

[BFJ03] B. Bui-Xuan, A. Ferreira, and A. Jarry. “Computing shortest, fastest, and foremost journeys in dynamic networks”. In: *International Journal of Foundations of Computer Science* 14.2 (2003), 267–285. DOI: 10.1142/S0129054103001728.

[Bur+03] C. Burch et al. “A Decomposition-Based Pseudoapproximation Algorithm for Network Flow Inhibition”. In: *Network Interdiction and Stochastic Integer Programming.* Ed. by D. L. Woodruff. Kluwer Academic Press, 2003. Chap. 1, 51–68. DOI: 10.1007/0-306-48109-X_3.

[BSR20] S. Busam, L. E. Schäfer, and S. Ruzika. *The two player shortest path network interdiction problem.* `http://arxiv.org/abs/2004.08338`. 2020.

[Cac+22a] V. Cacchiani et al. “Knapsack problems — An overview of recent advances. Part I: Single knapsack problems”. In: *Computers & Operations Research* 143 (2022), 105692. DOI: `10.1016/j.cor.2021.105692`.

[Cac+22b] V. Cacchiani et al. “Knapsack problems — An overview of recent advances. Part II: Multiple, multidimensional, and quadratic knapsack problems”. In: *Computers & Operations Research* 143 (2022), 105693. DOI: `10.1016/j.cor.2021.105693`.

[Cas+12] A. Casteigts et al. “Time-varying graphs and dynamic networks”. In: *International Journal of Parallel, Emergent and Distributed Systems* 27.5 (2012), 387–408. DOI: `10.1080/17445760.2012.668546`.

[Cas+21] A. Casteigts et al. “Finding Temporal Paths Under Waiting Time Constraints”. In: *Algorithmica* 83.9 (2021), 2754–2802. DOI: `10.1007/s00453-021-00831-w`.

[CM15] D. Che and L.W. Mays. “Development of an Optimization/Simulation Model for Real-Time Flood-Control Operation of River-Reservoirs Systems”. In: *Water Resources Management* 29 (2015), 3987–4005. DOI: `10.1007/s11269-015-1041-8`.

[CZ17] S. R. Chestnut and R. Zenklusen. “Hardness and approximation for network flow interdiction”. In: *Networks* 69.4 (2017), 378–387. DOI: `10.1002/net.21739`.

[Csi+91] J. Csirik et al. “Heuristics for the 0-1 Min-Knapsack Problem”. In: *Acta Cybernetica* 10.1-2 (1991), 15–20.

[DM11] C. D’Ambrosio and S. Martello. “Heuristic algorithms for the general nonlinear separable knapsack problem”. In: *Computers & Operations Research* 38.2 (2011), 505–513. DOI: `10.1016/j.cor.2010.07.010`.

[DMM18] C. D’Ambrosio, S. Martello, and L. Mencarelli. “Relaxations and heuristics for the multiple non-linear separable knapsack problem”. In: *Computers & Operations Research* 93 (2018), 79–89. DOI: `10.1016/j.cor.2017.12.017`.

[DAm+18] C. D’Ambrosio et al. “On the Product Knapsack Problem”. In: *Optimization Letters* 12.4 (2018), 691–712. DOI: `10.1007/s11590-017-1227-5`.

[DP22] A. Deligkas and I. Potapov. “Optimizing reachability sets in temporal graphs by delaying”. In: *Information and Computation* 285 (2022), 104890. DOI: `10.1016/j.ic.2022.104890`.

[Die97] R. Diestel. *Graph Theory.* Springer, 1997. DOI: `10.1007/978-3-662-53622-3`.

[Dur66] E. P. Durbin. *An Interdiction Model of Highway Transportation.* RM-4945-PR, RAND Corporation, Santa Monica, CA. 1966.

[Ehr05] M. Ehrgott. *Multicriteria Optimization.* Springer, 2005. DOI: `10.1007/3-540-27659-9`.

[EKN19] K. Elbassioni, A. Karapetyan, and T.T. Nguyen. “Approximation schemes for r-weighted Minimization Knapsack problems”. In: *Annals of Operations Research* 279.1-2 (2019), 367–386. DOI: `10.1007/s10479-018-3111-9`.

[EMS21] J. Enright, K. Meeks, and F. Skerman. “Assigning times to minimise reachability in temporal graphs”. In: *Journal of Computer and System Sciences* 115 (2021), 169–186. DOI: 10.1016/j.jcss.2020.08.001.

[Enr+21] J. Enright et al. “Deleting edges to restrict the size of an epidemic in temporal networks”. In: *Journal of Computer and System Sciences* 119 (2021), 60–77. DOI: 10.1016/j.jcss.2021.01.007.

[EFM07] A. Epstein, M. Feldman, and Y. Mansour. “Strong equilibrium in cost sharing connection games”. In: *Proceedings of the 8th ACM Conference on Electronic Commerce (EC)*. 2007, 84–92. DOI: 10.1145/1250910.1250924.

[EKP02] T. Erlebach, H. Kellerer, and U. Pferschy. “Approximating Multiobjective Knapsack Problems”. In: *Management Science* 48.12 (2002), 1603–1612. DOI: 10.1287/mnsc.48.12.1603.445.

[FS21] A. Fekete and S. Sandholz. “Here Comes the Flood, but Not Failure? Lessons to Learn after the Heavy Rain and Pluvial Floods in Germany 2021”. In: *Water* 13.21 (2021), 3016. DOI: 10.3390/w13213016.

[Fil14] T. Filatova. “Market-based instruments for flood risk management: A review of theory, practice and perspectives for climate adaptation policy”. In: *Environmental Science & Policy* 37 (2014), 227–242. DOI: 10.1016/j.envsci.2013.09.005.

[Flu+20] T. Fluschnik et al. “Temporal Graph Classes: A View Through Temporal Separators”. In: *Theoretical Computer Science* 806 (2020), 197–218. DOI: 10.1016/j.tcs.2019.03.031.

[FF62] L. R. Ford and D. R. Fulkerson. *Flows in Networks*. Princeton University Press, 1962. DOI: 10.1515/9781400875184.

[GJ79] M.R. Garey and D.S. Johnson. *Computers and intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., 1979.

[GL79] G.V. Gens and E.V. Levner. “Computational complexity of approximation algorithms for combinatorial problems”. In: *Proceedings of the 8th International Symposium on Mathematical Foundations of Computer Science (MFCS)*. Vol. 74. LNCS. 1979, 292–300. DOI: 10.1007/3-540-09526-8_26.

[Ger16] German Association for Water, Wastewater and Waste. *Merkblatt DWA-M 119, Risikomanagement in der kommunalen Überflutungsvorsorge für Entwässerungssysteme bei Starkregen (Risk management in municipal flood protection for drainage systems in the event of heavy rain)*. 2016.

[Gle+21] A. Gleixner et al. “MIPLIB 2017: data-driven compilation of the 6th mixed-integer programming library”. In: *Mathematical Programming Computation* 13.3 (2021), 443–490. DOI: 10.1007/s12532-020-00194-3.

[Gua22] The Guardian. *Pakistan floods: before-and-after images show extent of devastation*. https://www.theguardian.com/world/2022/aug/31/pakistan-floods-before-and-after-images-show-extent-of-devastation. accessed 22 March 2023. 2022.

[GJ00] M. M. Güntzer and D. Jungnickel. “Approximate minimization algorithms for the 0/1 Knapsack and Subset-Sum Problem”. In: *Operations Research Letters* 26.2 (2000), 55–66. DOI: 10.1016/S0167-6377(99)00066-8.

[Hal+19] N. Halman et al. “Bi-criteria Path Problem with Minimum Length and Maximum Survival Probability”. In: *OR Spectrum* 41 (2019), 469–489. DOI: 10.1007/s00291-018-0543-1.

[HRT21] A. Herzel, S. Ruzika, and C. Thielen. “Approximation Methods for Multiobjective Optimization Problems: A Survey”. In: *INFORMS Journal on Computing* 33.4 (2021), 1284–1299. DOI: 10.1287/ijoc.2020.1028.

[Hoc95] D. S. Hochbaum. “A nonlinear Knapsack problem”. In: *Operations Research Letters* 17.3 (1995), 103–110. DOI: 10.1016/0167-6377(95)00009-9.

[HKT17] M. Holzhauser, S.O. Krumke, and C. Thielen. “Maximum flows in generalized processing networks”. In: *Journal of Combinatorial Optimization* 33 (2017), 1226–1256. DOI: 10.1007/s10878-016-0031-y.

[HS21] T. Holzmann and J.C. Smith. “The Shortest Path Interdiction Problem with Randomized Interdiction Strategies: Complexity and Algorithms”. In: *Operations Research* 69.1 (2021), 82–99. DOI: 10.1287/opre.2020.2023.

[Hua+18] C.L. Huang et al. “Optimization of low impact development layout designs for megacity flood mitigation”. In: *Journal of Hydrology* 564 (2018), 542–558. DOI: 10.1016/j.jhydrol.2018.07.044.

[Ibi+22] A. Ibiapina et al. *Menger’s Theorem for Temporal Paths (Not Walks)*. https://arxiv.org/abs/2206.15251. 2022.

[Ins] Institut für technisch-wissenschaftliche Hydrologie GmbH. *HYSTEM-EXTRAN*. https://itwh.de/en/software-products/desktop/hystem-extran/. accessed 02 September 2022.

[IPC21] IPCC. *Climate Change 2021: The Physical Science Basis. Contribution of Working Group I to the Sixth Assessment Report of the Intergovernmental Panel on Climate Change*. V. Masson-Delmotte, P. Zhai, A. Pirani, S.L. Connors, C. Péan, S. Berger, N. Caud, Y. Chen, L. Goldfarb, M.I. Gomis, M. Huang, K. Leitzell, E. Lonnoy, J.B.R. Matthews, T.K. Maycock, T. Waterfield, O. Yelekçi, R. Yu, and B. Zhou (eds.), Cambridge University Press. 2021.

[JKS08] B.K. Jack, C. Kousky, and K.R.E. Sims. “Designing payments for ecosystem services: Lessons from previous experience with incentive-based mechanisms”. In: *Proceedings of the National Academy of Sciences (PNAS)* 105.28 (2008), 9465–9470. DOI: 10.1073/pnas.0705503104.

[Jin19] C. Jin. “An Improved FPTAS for 0-1 Knapsack”. In: *Proceedings of the 46th International Colloquium on Automata, Languages, and Programming (ICALP)*. Vol. 132. Leibniz International Proceedings in Informatics (LIPIcs). 2019, 76:1–76:14. DOI: 10.4230/LIPIcs.ICALP.2019.76.

[KN09] S. Kameshwaran and Y. Narahari. “Nonconvex piecewise linear knapsack problems”. In: *European Journal of Operational Research* 192.1 (2009), 56–68. doi: 10.1016/j.ejor.2007.08.044.

[KPP04] H. Kellerer, U. Pferschy, and D. Pisinger. *Knapsack Problems*. Springer, 2004. doi: 10.1007/978-3-540-24777-7.

[KKK00] D. Kempke, J. Kleinberg, and A. Kumar. “Connectivity and Inference Problems for Temporal Networks”. In: *Proceedings of the 32nd ACM Symposium on the Theory of Computing (STOC)*. 2000, 504–513. doi: 10.1006/jcss.2002.1829.

[Kha+08] L. Khachiyan et al. “On Short Paths Interdiction Problems: Total and Node-Wise Limited Interdiction”. In: *Theory of Computing Systems* 43.2 (2008), 204–233. doi: 10.1007/s00224-007-9025-6.

[KP21] S. Khalilpourazari and S. H. R. Pasandideh. “Designing emergency flood evacuation plans using robust optimization and artificial intelligence”. In: *Journal of Combinatorial Optimization* 41 (2021), 640–677. doi: 10.1007/s10878-021-00699-0.

[Kle+21] W.J. Klerk et al. “Optimal planning of flood defence system reinforcements using a greedy search algorithm”. In: *Reliability Engineering & System Safety* 207 (2021), 107344. doi: 10.1016/j.ress.2020.107344.

[Koc+22] T. Koch et al. “Progress in mathematical programming solvers from 2001 to 2020”. In: *EURO Journal on Computational Optimization* 10 (2022), 100031. doi: 10.1016/j.ejco.2022.100031.

[Koe83] Jacob Koene. “Minimal cost flow in processing networks: a primal approach”. PhD thesis. Centrum voor Wiskunde & Informatica, Amsterdam, 1983.

[Kov96] Y. M. Kovalyov. “A rounding technique to construct approximation algorithms for knapsack and partition type problems”. In: *Applied Mathematics and Computer Science* 6.4 (1996), 789–801.

[Kre+05] H. Kreibich et al. “Flood loss reduction of private households due to building precautionary measures–lessons learned from the Elbe flood in August 2002”. In: *Natural hazards and earth system sciences* 5.1 (2005), 117–126. doi: 10.5194/nhess-5-117-2005.

[Kru+98] S.O. Krumke et al. “Approximation Algorithms for Certain Network Improvement Problems”. In: *Journal of Combinatorial Optimization* 2.3 (1998), 257–288. doi: 10.1023/A:1009798010579.

[LSH94] M Labbé, E.F. Schmeichel, and S. L. Hakimi. “Approximation algorithms for the capacitated plant allocation problem”. In: *Operations Research Letters* 15.3 (1994), 115–126. doi: 10.1016/0167-6377(94)90046-9.

[LFH16] M.B. Lowry, P. Furth, and T. Hadden-Loh. “Prioritizing new bicycle facilities to improve low-stress network connectivity”. In: *Transportation Research Part A: Policy and Practice* 86 (2016), 124–140. doi: 10.1016/j.tra.2016.02.003.

[Maa+23] N. Maack et al. "On finding separators in temporal split and permutation graphs". In: *Journal of Computer and System Sciences* 135 (2023), 1–14. DOI: 10.1016/j.jcss.2023.01.004.

[MHJ18] J. Machac, T. Hartmann, and J. Jilkova. "Negotiating land for flood risk management: upstream-downstream in the light of economic game theory". In: *Journal of Flood Risk Management* 11.1 (2018), 66–75. DOI: 10.1111/jfr3.12317.

[MRS12] A. Malaviya, C. Rainwater, and T. Sharkey. "Multi-period network interdiction problems with applications to city-level drug enforcement". In: *IIE Transactions* 44.5 (2012), 368–380. DOI: 10.1080/0740817X.2011.602659.

[Man17] P. Manurangsi. "Almost-polynomial ratio ETH-hardness of approximating densest k-subgraph". In: *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*. 2017, 954–961. DOI: 10.1145/3055399.3055412.

[MWG95] A. Mas-Colell, M. D. Whinston, and J. R. Green. *Microeconomic Theory*. Oxford University Press, 1995.

[MS16] O. Michail and P.G. Spirakis. "Traveling salesman problems in temporal graphs". In: *Theoretical Computer Science* 634 (2016), 1–23. DOI: 10.1016/j.tcs.2016.04.006.

[Moh+23] S. Mohr et al. "A multi-disciplinary analysis of the exceptional flood event of July 2021 in central Europe – Part 1: Event description and analysis". In: *Natural Hazards and Earth System Sciences* 23.2 (2023), 525–551. DOI: 10.5194/nhess-23-525-2023.

[Mol20] H. Molter. "Classic Graph Problems Made Temporal – A Parameterized Complexity Analysis". PhD thesis. Technische Universität Berlin, 2020.

[Mol22] H. Molter. "The Complexity of Finding Temporal Separators under Waiting Time Constraints". In: *Information Processing Letters* 175 (2022), 106229. DOI: 10.1016/j.ipl.2021.106229.

[MRZ21] H. Molter, M. Renken, and P. Zschoche. *Temporal Reachability Minimization: Delaying vs. Deleting*. http://arxiv.org/abs/2102.10814. 2021.

[MPS22] M. Monaci, C. Pike-Burke, and A. Santini. "Exact algorithms for the 0-1 Time-Bomb Knapsack Problem". In: *Computers & Operations Research* 145 (2022), 105848. DOI: 10.1016/j.cor.2022.105848.

[Mun+21] H. S. Munawar et al. "An Integrated Approach for Post-Disaster Flood Management Via the Use of Cutting-Edge Technologies and UAVs: A Review". In: *Sustainability* 13.14 (2021), 7925. DOI: 10.3390/su13147925.

[MC09] L. Murawski and R.L. Church. "Improving accessibility to rural health services: The maximal covering network improvement problem". In: *Socio-Economic Planning Sciences* 43.2 (2009), 102–110. DOI: 10.1016/j.seps.2008.02.012.

[MMG07] A. T. Murray, T. C. Matisziw, and T. H. Grubesic. "Critical Network Infrastructure Analysis: Interdiction and System Flow". In: *Journal of Geographical Systems* 9 (2007), 103–117. DOI: 10.1007/s10109-006-0039-4.

[MO19] P. Mutzel and L. Oettershagen. “On the Enumeration of Bicriteria Temporal Paths”. In: *Proceedings of the 15th Annual Conference on Theory and Applications of Models of Computation (TAMC).* Vol. 11436. Lecture Notes in Computer Science. 2019, 518–535. DOI: 10.1007/978-3-030-14812-6_32.

[Nem+22] B. Nematollahi et al. “A Stochastic Conflict Resolution Optimization Model for Flood Management in Detention Basins: Application of Fuzzy Graph Model”. In: *Water* 14.5 (2022), 774. DOI: 10.3390/w14050774.

[New20] CBC News. *Kenya floods have killed nearly 200, displaced thousands.* https://www.cbc.ca/news/world/kenya-deadly-floods-1.5557400. accessed 22 March 2023. 2020.

[Ngo+16] T.T. Ngo et al. “Optimization of Upstream Detention Reservoir Facilities for Downstream Flood Mitigation in Urban Areas”. In: *Water* 8.7 (2016), 290. DOI: 10.3390/w8070290.

[Oet22] L. Oettershagen. “Temporal Graph Algorithms”. PhD thesis. Rheinische Friedrich-Wilhelms-Universität Bonn, 2022.

[Orl84] J. B. Orlin. “Minimum Convex Cost Dynamic Network Flows”. In: *Mathematics of Operations Research* 9.2 (1984), 190–207. DOI: 10.1287/moor.9.2.190.

[Orl13] J. B. Orlin. “Max Flows in $\mathcal{O}(nm)$ Time, or Better”. In: *Proceedings of the 45th ACM Symposium on the Theory of Computing (STOC).* 2013, 765–774. DOI: 10.1145/2488608.2488705.

[PR14] R.G. Parker and R.L. Rardin. *Discrete optimization.* Elsevier, 2014. DOI: 10.1016/C2009-0-22165-9.

[PS16] U. Pferschy and J. Schauer. “Approximation of the quadratic knapsack problem”. In: *INFORMS Journal on Computing* 28.2 (2016), 308–318. DOI: 10.1287/ijoc.2015.0678.

[PST21] U. Pferschy, J. Schauer, and C. Thielen. “Approximating the product knapsack problem”. In: *Optimization Letters* 15 (2021), 2529–2540. DOI: 10.1007/s11590-021-01760-x.

[Phi93] C. Phillips. “The Network Inhibition Problem”. In: *Proceedings of the 25th ACM Symposium on the Theory of Computing (STOC).* 1993, 776–785. DOI: 10.1145/167088.167286.

[PBA14] J.K. Poussin, W.J.W. Botzen, and J.C.J.H. Aerts. “Factors of influence on flood damage mitigation behaviour by households”. In: *Environmental Science & Policy* 40 (2014), 69–77. DOI: 10.1016/j.envsci.2014.01.013.

[RS17] J. Rajczak and C. Schär. “Projections of Future Precipitation Extremes Over Europe: A Multimodel Assessment of Climate Simulations”. In: *Journal of Geophysical Research: Atmospheres* 122.20 (2017), 10773–10800. DOI: 10.1002/2017JD027176.

[Roe05] Gerard H Roe. "Orographic precipitation". In: *Annual Review of earth and planetary sciences* 33.1 (2005), 645–671. DOI: 10.1146/annurev.earth.33.092203.122541.

[RW07] J. O. Royset and R. K. Wood. "Solving the Bi-Objective Maximum-Flow Network-Interdiction Problem". In: *INFORMS Journal on Computing* 19.2 (2007), 175–184. DOI: 10.1287/ijoc.1060.0191.

[Sch+20] L. E. Schäfer et al. "The Bicriterion Maximum Flow Network Interdiction Problem in s-t-Planar Graphs". In: *Operations Research Proceedings 2019: Selected Papers of the Annual International Conference of the German Operations Research Society.* Springer, 2020, 133–139. DOI: 10.1007/978-3-030-48439-2_16.

[Sch+14] T. G. Schmitt et al. "An Optimization and Decision Support Tool for Long-Term Strategies in the Transformation of Urban Water Infrastructure". In: *Proceedings of the 11th International Conference on Hydroinformatics (HIC).* 2014, 1–8.

[Sch98] A. Schrijver. *Theory of Linear and Integer Programming.* John Wiley & Sons, 1998.

[SS16] J. A. Sefair and J. C. Smith. "Dynamic shortest-path interdiction". In: *Networks* 68.4 (2016), 315–330. DOI: 10.1002/net.21712.

[Sie18] T. Siekmann. *Methodik zur Priorisierung von Maßnahmen der Sturzflutvorsorge.* https://www.siekmann-ingenieure.de/media/priorisierung-massnahmen_methodik.pdf. accessed 02 September 2022. 2018.

[SS20] J.C. Smith and Y. Song. "A survey of network interdiction models and algorithms". In: *European Journal of Operational Research* 283.3 (2020), 797–811. DOI: 10.1016/j.ejor.2019.06.024.

[SL06] W.Y. Szeto and H.K. Lo. "Transportation network improvement and tolling strategies: The issue of intergeneration equity". In: *Transportation Research Part A: Policy and Practice* 40.3 (2006), 227–243. DOI: 10.1016/j.tra.2005.06.004.

[Tas21] Byron Tasseff. "Optimization of Critical Infrastructure with Fluids". PhD thesis. University of Michigan, 2021.

[Tay16] R. Taylor. "Approximation of the Quadratic Knapsack Problem". In: *Operations Research Letters* 44.4 (2016), 495–497. DOI: 10.1016/j.orl.2016.05.005.

[VTL82] J. Valdes, R. E. Tarjan, and E. L. Lawler. "The recognition of Series Parallel digraphs". In: *SIAM Journal on Computing* 11.2 (1982), 298–313. DOI: 10.1145/800135.804393.

[Vaz01] V. V. Vazirani. *Approximation Algorithms.* Vol. 1. Springer, 2001. DOI: 10.1007/978-3-662-04565-7.

[WH08] C.C. Wei and N.S. Hsu. "Multireservoir real-time operations for flood control using balanced water level index method". In: *Journal of Environmental Management* 88.4 (2008), 1624–1639. DOI: 10.1016/j.jenvman.2007.08.004.

[WS11] D.P. Williamson and D.B. Shmoys. *The Design of Approximation Algorithms.* Cambridge university press, 2011. DOI: 10.1017/CBO9780511921735.

[Woe00] G. J. Woeginger. "When Does a Dynamic Programming Formulation Guarantee the Existence of a Fully Polynomial Time Approximation Scheme (FPTAS)?" In: *INFORMS Journal on Computing* 12.1 (2000), 57–74. DOI: 10.1287/ijoc.12.1.57.11901.

[Woj18] D. Wojtczak. "On strong NP-completeness of rational problems". In: *Proceedings of the 13th International Computer Science Symposium in Russia (CSR).* Vol. 10846. LNCS. 2018, 308–320. DOI: 10.1007/978-3-319-90530-3_26.

[Wol64] R. D. Wollmer. "Removing Arcs from a Network". In: *Operations Research* 12.6 (1964), 934–940. DOI: 10.1287/opre.12.6.934.

[Wol70] R. D. Wollmer. "Algorithms for Targeting Strikes in a Lines-of-Communication Network". In: *Operations Research* 18.3 (1970), 497–515. DOI: 10.1287/opre.18.3.497.

[Woo93] R. K. Wood. "Deterministic Network Interdiction". In: *Mathematical and Computer Modelling* 17.2 (1993), 1–18. DOI: 10.1016/0895-7177(93)90236-R.

[WKG14] M. Woodward, Z. Kapelan, and B. Gouldby. "Adaptive Flood Risk Management Under Climate Change Uncertainty Using Real Options And Optimization". In: *Risk Analysis* 34.1 (2014), 75–92. DOI: 10.1111/risa.12088.

[Wu+16] H. Wu et al. "Efficient Algorithms for Temporal Path Computation". In: *IEEE Transactions on Knowledge and Data Engineering* 28.11 (2016), 2927–2942. DOI: 10.1109/TKDE.2016.2594065.

[Zsc+20] P. Zschoche et al. "The complexity of finding small separators in temporal graphs". In: *Journal of Computer and System Sciences* 107 (2020), 72–92. DOI: 10.1016/j.jcss.2019.07.006.

[ZVH18] P. Zwaneveld, G. Verweij, and S. van Hoesel. "Safe dike heights at minimal costs: An integer programming approach". In: *European Journal of Operational Research* 270.1 (2018), 294–301. DOI: 10.1016/j.ejor.2018.03.012.