APPROXIMATION ALGORITHMS FOR MULTISTAGE SUBGRAPH PROBLEMS

DISSERTATION

zur Erlangung des Grades eines

Doktors der Naturwissenschaften

im Fachbereich Mathematik/Informatik/Physik der

Universität Osnabrück

vorgelegt von

NIKLAS JOHANNES TROOST

Osnabrück

2023

Dekan:	**Prof. Dr. Tim Römer** Universität Osnabrück
Gutachter:	**Prof. Dr. Markus Chimani** Universität Osnabrück **Prof. Dr. Christian Komusiewicz** Friedrich-Schiller-Universität Jena

ABSTRACT

The concepts of classical graph theory offer a framework for modeling problems from almost all areas of life. Temporal graphs in particular provide a way to represent and analyze dynamic data, enabling algorithmic solutions to problems dealing with structures that change over time. For the extension to time-related data, different modifications of the graph concept can be useful, depending on the optimization goal. In this work, the focus is on the analysis of problems on finite sequences of graphs whose sequence members are to be read as snapshots of the same graph at different points in time.

The key topic is to solve a given combinatorial optimization problem on a sequence of graphs such that the resulting solution sequence satisfies two conditions: (i) each solution is optimal for its respective graph and (ii) solutions of successive graphs are as similar as possible. Problems of this abstract form are called *multistage problems*, since their instances consist of multiple *stages*.

First, the classical assignment problem Maximum Matching is exemplarily transferred into such a multistage setting. For the two resulting problems MIM and MUM various approximation algorithms and reductions are developed. The algorithmic approaches for MIM can be generalized to apply to a broad class of multistage formulations of classical problems in graph theory. This general class of problems on multistage graphs, the so-called *Multistage Subgraph Problems*, is the focus of the second part of this thesis. After discussing the corresponding definitions and adapted algorithms, we present numerous examples of MSPs, where these adapted algorithms can be applied. As we also study the complexity of each example, this provides an overview of the similarities and differences of such multistage problems. In a third part, the performance of the approximation algorithms in practice is investigated by applying them to example instances for the Multistage Shortest Path problem. A comparison is made with derived heuristics and an exact (but slow) algorithm, investigating their respective solution quality and running time. Additional problem-specific results on Multistage Shortest Path and Multistage Minimum Weight Spanning Tree complete the thesis.

ZUSAMMENFASSUNG

Die Konzepte der Graphentheorie ermöglichen die Modellierung von Problemstellungen aus nahezu allen Lebensbereichen. Insbesondere lassen sich mit der Darstellung dynamischer Daten als temporale Graphen Probleme darstellen, analysieren und algorithmisch lösen, die auf zeitveränderlichen Strukturen basieren. Für die Erweiterung auf zeitbezogene Daten können, abhängig vom Optimierungsziel, verschiedene Modifikationen des Graphenkonzept sinnvoll sein. In dieser Arbeit liegt der Fokus auf der Analyse von Problemen auf endlichen Folgen von Graphen, deren Folgenglieder zu lesen sind als Momentaufnahmen desselben Graphen zu unterschiedlichen Zeitpunkten.

Das Kernproblem lautet, ein gegebenes kombinatorisches Optimierungsproblem auf einer Folge von Graphen so zu lösen, dass die resultierende Folge von Lösungen zwei Bedingungen entspricht: (i) Jede Lösung ist optimal für ihren jeweiligen Graphen und (ii) Lösungen von aufeinanderfolgenden Graphen sind sich möglichst ähnlich. Probleme dieser abstrakten Form werden, da ihre Instanzen aus mehreren aufeinanderfolgenden *Stufen* bestehen, als *Multistage-Probleme* bezeichnet.

Zunächst wird exemplarisch das klassische Zuordnungsproblem Maximum Matching in ein solches Multistage-Setting übertragen. Für die zwei daraus resultierenden Problemstellungen MIM und MUM werden diverse Approximationsalgorithmen und Reduktionen entwickelt. Die algorithmischen Herangehensweisen für MIM lassen sich verallgemeinern zur Anwendung auf eine breite Klasse an Multistage-Formulierungen von klassischen Problemen der Graphentheorie. Diese allgemeinen Problemklasse, die sogenannten *Multistage-Teilgraph-Probleme*, stehen im Fokus des zweiten Teils dieser Arbeit. Nach einer Diskussion der entsprechenden Definitionen und dem Anpassen der Algorithmen werden zahlreiche Beispiele für solche Problemstellungen vorgestellt, für die diese anwendbar sind. Indem auch die Komplexität dieser Beispiele untersucht wird, entsteht ein Überblick über die Gemeinsamkeiten und Unterschiede solcher Multistage-Problemstellungen. In einem dritten Teil wird die Performanz der Approximationsalgorithmen in der Praxis untersucht, indem diese auf Beispielinstanzen für das Problem Multistage Shortest Path angewendet werden. Im Vergleich mit abgeleiteten Heuristiken und einem exakten (aber langsamen) Algorithmus erfolgt eine Einordnung in Bezug auf Lösungsgüte und Laufzeit. Ergänzende problemspezifische Ergebnisse zu Multistage Shortest Path und Multistage Minimum Weight Spanning Tree schließen die Arbeit ab.

DANKSAGUNGEN

Mein Dank gebührt den vielen Einzelpersonen und Menschengruppen, die mich über die letzten Jahre begleitet und bestärkt haben – sowohl durch direkte Unterstützung, als auch durch unsere gemeinsamen Aktivitäten, welche die (zwangsweise auftretenden) frustrierenden Zeiten wissenschaftlicher Flaute immer wieder mit Erfolgen ausfüllen konnten.

Zuerst möchte ich meinen Betreuer Markus Chimani nennen, mit dem die Zusammenarbeit nicht nur auf fachlicher, sondern auch auf zwischenmenschlicher Ebene immer besonders angenehm war – ob im Büro oder im Garten, auf der Couch oder auf dem Kinderspielplatz. Vor allem unsere gemeinsame Arbeit an der Entwicklung der Lehrveranstaltungen hat mir immer große Freude bereitet. Darüber hinaus bin ich dir sehr dankbar für deine Offenheit, dir mit mir den neuen Themenbereich der Multistage-Probleme gemeinsam zu erschließen.

Ich danke auch meinen lieben aktuellen und ehemaligen Kollegen der *AG Theoretische Informatik*; neben den unzählbaren Stunden an wissenschaftlicher Arbeit, technischer Unterstützung und natürlich auch Ablenkung, die ich von euch einfordern durfte:

Tilo für seine beharrlichen Rückfragen und das Talent, genau im richtigen Momenten eine Pause auszurufen; Stephan für ein immer offenes Ohr, viel Schokolade und eine inspirierende Bürogemeinschaft; Alex für das geteilte Schicksal der „Heimat in der Fremde“, sowohl regional als auch fachlich; Fritz für kontinuierliche Kaffeeversorgung, tiefste Fachkenntnis und Wissbegierde in allen Lebenslagen; Max für unvergessliche gemeinsame Besuche der *Frico*-Konferenz und das Korrekturlesen dieser Arbeit; Mirko für so viele geteilte Interessen und seinen hartnäckigen Überschwang in den Momenten, in denen er mir selbst fehlte. Auch unseren (ehemaligen) studentischen Hilfskräften möchte ich danken für verlässliche gemeinsame Arbeit in der Lehre, technischen Support und zahlreiche gewinnbringende fachliche wie private Diskussionen.

Bedanken möchte ich mich auch bei all den Gruppierungen, die mir einen wunderbaren Ausgleich zur wissenschaftlichen Arbeit geboten haben – mit euch den Kopf auf Knopfdruck ausschalten zu können empfinde ich als großes Privileg. Das sind: die *KjG St. Barbara*, die *Musicalschule Hans & Alice*, das *Uniorchester Osnabrück*, die *Blue Sundays* und *No risk, no funk*.

Zu guter Letzt stehen natürlich und glücklicherweise viele Freunde und meine Familie; Gudrun und Michael, meine Geschwister mit ihren Familien, Oma und meine Eltern, die mich alle auf die eine odere andere Weise unterstützt und nie an mir gezweifelt haben.

Ganz besonders danke ich Natascha, die mir zu jeder Zeit beistand und ohne die ich diesen Lebensabschnitt nicht im Entferntesten als eine so glückliche und erfüllende Zeit in Erinnerung behalten könnte. Danke!

CONTENTS

1

INTRODUCTION

As you discover
changing world
You can't be guessing
You must be for sure
— Earth, Wind & Fire: *The Changing Times* [EWF81]

Most disciplines in science and engineering devise models of the world to better understand it. One of the most successful models in computer science is that of *graphs*, capturing objects as *vertices* and relationships between pairs of objects as *edges* connecting those vertices. Put simply, a graph is a mathematical model of a network-like structure, providing plentiful applications. A graph can model, for example,

- a group of people sharing contact data,
- train stations and tracks connecting them,
- computers that communicate via network cables,
- protein structures that are related.

However, as many real world situations are subject to change over time, a static model is often not sufficient and a dynamic model is needed to reflect those changes. Natural examples for graphs that vary over time are communication networks where the vertices represent cell phones and cell phone towers, or a set of mobile agents such as drones or robots. If two vertices are close enough to communicate directly, the graph contains an edge between those vertices. But since the agents are moving independently, the graph is continuously changing whenever two vertices lose contact or establish new connections. Figure 1.1 shows an example of such a graph.

In this chapter, we give a brief overview of various ways to enrich graph models with temporal data and the problems they are typically associated with. We continue with a more detailed description of the class of *multistage problems* and introduce some notation that will be used throughout this thesis.

1.1 TEMPORAL GRAPHS

In general, every aspect of a graph may be subject to change: edges or even vertices may appear or disappear, and also edge weights (or vertex weights, if

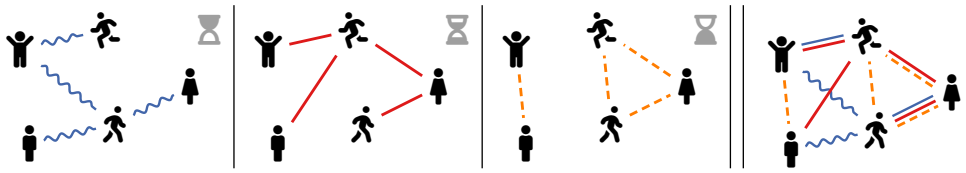


Figure 1.1: An example of a temporal graph, where each vertex represents a person and edges change over time. The first three graphs each show a snapshot of the temporal graph at three specified points in time. The fourth graph shows the whole temporal graph at a glance: each edge is stylized according to the point in time at which it occurs; multiple occurrences are displayed by parallel edges. This is the representation of temporal graphs that we will mainly use in the following.

there are any) may have time-dependent numerical values. To avoid confusion, it is typically understood that the vertices have a unique identifier that is consistent over the whole *lifetime* of a temporal graph. Edges, however, may be treated differently, depending on the exact problem description. For example, in [KKK00; CFQ+12] each edge is equipped with a list of timestamps denoting the points in time at which the edge is present in the graph. Some models use real-valued time intervals [AMS+21], some consider periodic patterns [FMS09], some use draw a random graph at each point in time [CRR+22], and the common model in the long established field of *dynamic graph theory* is to consider a stream of update information encoding incremental changes of the graph [ST81; Epp91; HHS22]. Since we do not aim for algorithms with highly efficient running time or space usage in this thesis, we will model temporal graphs in their most basic form as *multistage graphs*[1], where a full copy of the current graph is explicitly given for each point in time (see Section 1.3 for specific definitions). Casteigts [Cas18] provides a survey on various temporal graph representations. Other good introductions showcasing the variety in temporal graph research are the works of Holme and Saramäki [HS19] and Michail [Mic16].

As rich in variety as models and encodings of temporal graphs are, so are the problem types that are typically considered. Some questions arise only by examining the specific structures of temporal graphs, e.g., determining temporal graph parameters [BM23; CFQ+12], finding recurring patterns of subgraphs [PBL17], identifying a subsequence of graphs with a generic graph property [BKK+19] or designing graphs that have some desired temporal property [AGM+17; MMS19]. Other questions are akin to classical problems from graph theory, but introduce new temporal constraints. When tasked to find an optimal path[2] between two vertices, the challenge is that edges may appear or

1 The term *multistage graph* is also sometimes used for *leveled* graphs, whose vertices are partitioned into levels, and edges join consecutive levels, see, e.g., [CLX+21]. That definition and results thereon are unrelated to our scenario.

2 Depending on the circumstances one might want to minimize the time between departure and arrival, or the time spent traveling (i.e. ignoring waiting times between routes); others might prefer a path with latest departure or earliest arrival.

disappear during the time it takes to travel from one vertex to the next. This gives rise to various definitions of temporal reachability, temporal distance and different notions of temporal paths or *journeys* [WCH+14; KKK00; CCS22; CPS21; Oet22; XFJ03].

Many other temporal graph problems are concerned with generalizations of classical graph problems like Vertex Cover or Matching. In dynamic graph theory [HHS22], the typical goal is to—possibly after each graph modification—update a solution (online) as quickly as possible, not necessarily caring about the specific amount of change made to the solution. In other (offline) problem situations, solutions are allowed to "spread out" along the time dimension, i.e., a single solution is not restricted to a point in time but covers a certain time window. For example, the Temporal Matching problem [BBR20] asks for a set of *time-edges* (i.e., an instance of an edge at a single point in time) such that in each time window of a given size, no two selected time-edges share a vertex. Other examples of similar problems can be found, e.g., in [Mol20; AMS+20]. A third flavor of temporal graph problems are so-called *multistage problems*. As they are the main focus of this thesis, they deserve a more detailed introduction.

1.2 MULTISTAGE PROBLEMS

In multistage problems (first introduced by Gupta, Talwar, and Wieder [GTW14], and Eisenstat, Mathieu, and Schabanel [EMS14]), we consider a multistage graph as a sequence of graphs, where each *stage*, i.e., each member of the sequence, represents the respective graph at a distinct point in time. The goal is to solve some classical graph problem independently for each stage, resulting in a sequence of solutions. Such problems arise, e.g., when a certain task has to be performed multiple times at discrete points in time, but the underlying graph instance has received several modifications between two such time points. Most importantly, additionally to the original problem's objective per stage, the *stage-wise objective*, in multistage problems we also aim to maximize the similarity between the individual stages' solutions—the *transition quality*. The challenge is to find solutions that are both "good" when viewed locally and work well together from a global perspective. In general, one may not want to yield the highest possible transition quality between two stages if this would cause an exorbitant loss of quality in other stage transitions.

Having two distinct optimization goals at hand, one typically considers a weighted sum of both measures to allow trade-offs between the quality of the individual solutions and the similarity of those solutions [GTW14; EMS14; HHK+21; BEK21; BEL+18; BES+21; BET22; FNR+22]. More formally, the typical objective is to optimize a combined quantity $O + T$ that measures both the objective value O of the individual per-stage solutions and the quality T of the transitions between subsequent solutions.[3] For example, consider a multistage variant of the Maximum Matching problem: Given a sequence of τ

3 Although most works aim for transitions where subsequent solutions should be similar, the inverse objective of pursuing *dissimilar* transitions has also been studied [FNS+20; KRZ21].

graphs $(G_i)_{i=1}^{\tau}$, find a matching M_i for each G_i such that $O + T$ is maximized. Here, $O := \sum_{i=1}^{\tau} |M_i|$ is the sum of the individual matchings' cardinalities and $T := \sum_{i=1}^{\tau-1} |M_i \cap M_{i+1}|$ is the sum of transition qualities measured as the cardinality of edges that are common between subsequent solutions.

Interestingly, most polynomial-time solvable graph problems (such as shortest paths, matchings, minimum cuts, etc.) yield NP-complete problems in a multistage setting: this often already occurs when only two stages are considered, and independent on whether one restricts themselves to optimal solutions per stage or not [Flu21; FNS+20; GTW14; HHK+21]. There is some work on identifying parameters that allow for fixed-parameter tractability of NP-hard multistage problems [FNS+20; BFK22; FNR+22; Flu21; FK22; HHK+21].

Another popular approach to tackle such problems are approximation algorithms [GTW14; BEK21; BEL+18; BES+21; BET22]. In an approximation setting, the combined objective allows to trade suboptimal transitions for suboptimal solutions in some stages. This is exploited, e.g., in a 2-approximation for a multistage Vertex Cover problem [BEK21] and a 3-approximation for a 3-stage Minimum Weight Perfect Matching problem on metric graphs [BEL+18]. In an online algorithm setting, the authors of [BES+21] show several upper and lower bounds for the competitive ratios of online algorithms considering a general type of *multistage subset maximization problems*; however, their algorithms are not considering running times and depend on polynomial oracles for the underlying single-stage problems.

To the best of our knowledge, all approximation results in this setting discuss the combined objective function that reflects a trade-off between the quality of each individual solution and the cost of the change over time (see, e.g., [GTW14; BEL+18]). However, this is a drawback if one requires each stage's solution to attain a certain quality guarantee, such as optimality.

OPTIMAL SOLUTIONS. Sometimes it is desired to guarantee optimal solutions in each stage, i.e., fixing O to its maximum value. Then the sole goal is to maximize T by picking a suitable optimal solution (out of the set of possible optimal solutions) per stage.

A natural approach to ensure this would be to adjust the weighting—e.g., by multiplying O with some large constant λ—such that the stage-wise objective O would contribute significantly more to the global objective than the transition quality T. However, considering approximation algorithms, this technique has a clear drawback. If λ is sufficiently large, an approximation algorithm would not need to consider the transition qualities at all, because it may achieve a large enough objective value from maximizing the stage-wise objectives. If on the other hand λ is only slightly smaller than necessary, there could be no guarantee that the solution would not sacrifice some optimal solutions for large gains in the transition quality.

Contrasting the combined objective, the focus of this thesis is to break up the interdependency between O and T for several types of multistage problems and require O to be optimal. Hence, any feasible solution must necessarily consist

of optimal solutions for each individual stage. Regarding exact algorithms, this problem type can be treated as a special case of the respective problem with combined objective, where O is scaled up as discussed above. However, approximation guarantees for the combined-objective setting are in general *not* transferable to this special case as the approximation may require non-optimal solutions in individual stages; we will see a detailed example in Section 2.3. Thus, approximation guarantees for this problem type provide a better measure for the difficulty in approximating the temporal transition cost. Although the concept of fixing O to optimality is novel, there exist some results on multistage problems where O is a constant due to the underlying single-stage problem. In particular, MULTISTAGE 2-SAT [Flu21] and MULTISTAGE 2-COLORING [FK22] have been studied from a parameterized complexity viewpoint.

1.3 DEFINITIONS AND NOTATION

For $n, n' \in \mathbb{Z}$, let $[n, n'] \coloneqq \{n, n+1, \dots, n'\} \subset \mathbb{Z}$ denote the integer range from n to n'. As we are mostly considering natural numbers, we will make extensive use of the shorthand notations $[\![n]\!] \coloneqq [0, n]$ and $[n] \coloneqq [1, n]$. An indexed sequence of n objects $(x_1, x_2, \dots, x_n)$ is thus concisely denoted as $(x_i)_{i \in [n]}$. For some logical statement φ, the *indicator function* $\mathbb{1}(\varphi)$ returns 1 if φ is satisfied and 0 otherwise.

GRAPHS. For a graph $G = (V, E)$, we refer to its vertices $V(G) \coloneqq V$ and edges $E(G) \coloneqq E$ collectively as *elements* $X(G) \coloneqq V \cup E$. For a set $W \subseteq V$ of vertices, let $\delta(W) \coloneqq \{uv \in E \mid u \in W, v \in V \setminus W\}$ denote the set of its *cut edges*; for a singleton $\{v\}$, we may write $\delta(v)$ instead of $\delta(\{v\})$. For a set $F \subseteq E$ of edges, let $V(F) \coloneqq \{v \in V \mid \delta(v) \cap F \neq \varnothing\}$ denote its incident vertices. For a vertex $v \in V$, $|\delta(v)|$ is the *degree of* v.

A *k-path* is a connected graph with k edges where exactly two vertices have degree 1 and all other vertices have degree 2. A *k-cycle* is a connected graph with k edges where every vertex has degree 2. A k-cycle is odd (even) if k is odd (even, respectively). A graph that does not contain any odd cycle as a subgraph is *bipartite*.

MULTISTAGE GRAPHS. A *multistage graph* is a finite sequence of $\tau > 1$ graphs $\mathcal{G} = (G_i)_{i \in [\tau]}$, where the vertices of each member are chosen from a common vertex superset $V(\mathcal{G})$. The graph $G_i =: (V_i, E_i)$ is the *ith stage of $\mathcal{G}$* (sometimes also referred to as a *snapshot*). The number τ of stages is sometimes called the *lifetime* of $\mathcal{G}$; if τ is assumed constant, $\mathcal{G}$ is a *τ-stage graph*.

Considering some fixed multistage graph, $E_\cap \coloneqq \bigcap_{i \in [\tau]} E_i$ denotes the set of edges that are common to all stages and $E_\cup \coloneqq \bigcup_{i \in [\tau]} E_i$ the entirety of its edges. The graph $G_\cup \coloneqq (V(E_\cup), E_\cup)$ is the *union graph* of $\mathcal{G}$ (sometimes also called the *underlying graph*). Given a multistage graph, its *intertwinement* $\chi(\mathcal{G}) \coloneqq \max_{i \in [\tau-1]} |E_i \cap E_{i+1}|$ is the maximum commonality between the edge sets of two subsequent stages. It thus provides a coarse measure for the similarity between the stages of a multistage graph.

APPROXIMATION. An approximation algorithm $\mathcal{A}$, or simply *approximation*, outputs a feasible, but not necessarily optimal solution for some optimization problem $\mathcal{P}$ in polynomial time. Considering an instance I of $\mathcal{P}$, $\mathsf{opt}(I)$ denotes the optimal objective value w.r.t. I and $\mathsf{apx}(I)$ the objective value that $\mathcal{A}$ computes on input I. If $\mathcal{P}$ has a maximization (minimization) goal, the approximation ratio of $\mathcal{A}$ is the infimum (supremum, respectively) of $\mathsf{apx}(I)/\mathsf{opt}(I)$ over all instances I of $\mathcal{P}$.

1.4 ORGANIZATION AND ORIGINAL PUBLICATIONS

This thesis is structured as follows. In Part I we develop several approximation results for two multistage matching variants. It is based on joint work with Markus Chimani and Tilo Wiedera that has been published in the conference proceedings of the *32nd International Workshop on Combinatorial Algorithms* (IWOCA 2021) [CTW21] and in a special issue of *Algorithmica* [CTW22].

In Part II, some of the previous findings are generalized to a broad class of similar multistage problems. We define the necessary notational machinery, reiterate the generalized results, apply them to some example problems, and discuss the NP-hardness of said examples. This part is based on joint work with Markus Chimani and Tilo Wiedera that has been accepted for publication in the conference proceedings of the *XII. Latin-American Algorithms, Graphs and Optimization Symposium* (LAGOS 2023).

The practical value of this approach is studied in Part III by conducting algorithmic experiments using the example problem of finding a shortest s-t-path. A main focus of this part is the work on identifying suitable instances and their discussion. While the problem is not directly motivated from practice, the experiments do provide some foundation for further theoretical research directions. These results are based on joint work with Markus Chimani that has been published in the conference proceedings of the *2nd Symposium on Algorithmic Foundations of Dynamic Networks* (SAND 2023) [CT23].

Finally, in Part IV, we give a brief overview of the main findings of the previous parts, present further, yet unpublished results on closely related problems and tie everything together by discussing some open questions.

Part I

MULTISTAGE PERFECT MATCHINGS

The good news? It's just a matter of time before you make a connection with someone new.
— Tinder FAQ: *Problems with Matches* [Tin23]

2

BACKGROUND

This part is based on joint work with Markus Chimani and Tilo Wiedera that has been published in the conference proceedings of the *32nd International Workshop on Combinatorial Algorithms* (IWOCA 2021) [CTW21] and in a special issue of the *Algorithmica* journal [CTW22].

A classical problem in computer science is that of computing a matching, i.e., a collection of edges that are independent in the sense that they do not share any vertices. While there are many practical applications, the problem is also well-known for its significant impact on research in theoretical computer science [LP86]. Even before being investigated in one of the first publications on multistage problems [GTW14], time-dependent variants of the traditional matching problems (perfect matching, maximum weight matching, etc.) have been considered in various disciplines, see Section 2.2. From a complexity point of view, the problem is particularly interesting as optimality for a single stage would be obtainable in polynomial time, but all known multistage variants are NP-hard already for two stages. There are several known approximation algorithms for multistage matching problems [BEL+18]; however, they all follow the trade-off paradigm. In this part of the thesis, we are concerned with maintaining a *perfect* matching on a multistage graph, such that the changes between consecutive matchings are minimized. More precisely, we aim to maximize the intersections, or minimize the unions between consecutive matchings.

We show that these problems are NP-hard even in very restricted scenarios. As our main contribution, we present the first non-trivial approximation algorithms for these problems: On the one hand, we devise a tight approximation on 2-stage graphs. On the other hand, we propose general methods to deduce multistage approximations from blackbox approximations on 2-stage graphs. After showcasing the complexities of our problems (Chapter 3), we will devise efficient approximation algorithms (Chapter 4).

2.1 DEFINITIONS AND PRELIMINARIES

A set $M \subseteq E$ of edges is a *matching* if every vertex is incident to at most one edge of M; it is a *perfect* matching if $|\delta(v) \cap M| = 1$ for every $v \in V$. Considering a graph with positive edge weights $w\colon E \to \mathbb{N}_{>0}$, a matching M has *maximum weight* if there is no matching M' with $\sum_{e\in M'} w(e) > \sum_{e\in M} w(e)$.

A *multistage perfect matching* in a multistage graph $\mathcal{G} = (G_1, \dots, G_\tau)$ is a sequence $\mathcal{M} := (M_i)_{i \in [\tau]}$ such that for each $i \in [\tau]$, M_i is a perfect matching in stage G_i. A multistage graph is *spanning* if $V_i = V$ for each $i \in [\tau]$. Let $n_i := |V_i|$ denote the number of vertices in stage $i \in [\tau]$, and $n := |V(E_\cup)|$ the number of vertices in the union graph.

All problems considered in this part (MIM, MUM, MinMPM, MaxMPM; see below) are of the following form: Given a multistage graph $\mathcal{G}$, we ask for a multistage perfect matching $\mathcal{M}$ optimizing some objective function. In their respective decision variants, the input furthermore consists of some value κ and we ask whether there is an $\mathcal{M}$ with objective value at most (minimization problems) or at least (maximization problems) κ.

Definition 1 (MIM and MIM$|_\tau$)**.** *Given a multistage graph $\mathcal{G}$, the* multistage intersection matching problem (MIM) *asks for a multistage perfect matching $\mathcal{M}$ of $\mathcal{G}$ with maximum* profit $p(\mathcal{M}) := \sum_{i \in [\tau-1]} |M_i \cap M_{i+1}|$*. If there is an upper bound t on the number of stages τ, we denote the problem by* MIM$|_t$*.*

We also consider the natural inverse objective, i.e., minimizing the unions. While the problems differ in the precise objective function, an optimal solution of MIM is optimal for MUM as well, and vice versa.

Definition 2 (MUM and MUM$|_\tau$)**.** *Given a multistage graph $\mathcal{G}$, the* multistage union matching problem (MUM) *asks for a multistage perfect matching $\mathcal{M}$ of $\mathcal{G}$ with minimum* cost $c(\mathcal{M}) := \sum_{i \in [\tau-1]} |M_i \cup M_{i+1}|$*. If there is an upper bound t on the number of stages τ, we denote the problem by* MUM$|_t$*.*

2.2 RELATED WORK

Some matching problems on temporal graphs are defined considering time windows and allow for parameterized complexity results [MMN+20; BBR20]. While the definitions ensure that there are no conflicting edges in each stage, the per-stage solutions are not necessarily perfect matchings. The problem of finding the largest edge set that induces a matching in each stage, which is shown to be W[1]-hard in [HHK+21], suffers from the same flaw.

In dynamic graph theory, a typical approach to tackle matchings is to make local changes to a previous solution, working through the stages one after another [BS15; BHI18; BLS+14; San07]. Naturally, this setting does not take into account which changes might give a benefit with respect to future graph stages and are thus not suited well for our problem definitions.

In the literature we find the problem MaxMPM [BEL+18], where the graph is augmented with time-dependent edge weights, and we want to maximize the value of each individual perfect matching (subject to the given edge costs) *plus* the total intersection profit. Our problem MIM is the special case where all edge costs are uniform, i.e., we only care about the multistage properties of the solution, as long as each stage is perfectly matched. There is also the inverse optimization problem MinMPM, where we minimize the value of each perfect

matching *plus* the number of matching edges newly introduced in each stage. We have APX-hardness for MaxMPM and MinMPM [BEL+18; GTW14] (for MinMPM one may assume a complete graph at each stage, possibly including edges of infinite weight). The latter remains APX-hard even for spanning 2-stage graphs with bipartite union graph and no edge weights (i.e., we only minimize the number of edge swaps) [BEL+18]. For uniform edge weights, the objective of MinMPM is to minimize $\sum_{i\in[\tau-1]} |M_{i+1} \setminus M_i|$, which is similar but slightly different to MUM (equal up to additive $\sum_{i\in[\tau-1]} n_i/2$).

When restricting MaxMPM and MinMPM to uniform edge weights, optimal solutions for MIM, MUM, MaxMPM, and MinMPM are identical (although they yield different objective value); thus MIM and MUM are NP-hard as well. However, the APX-hardness of MinMPM does not imply APX-hardness of MUM since their objective functions differ. Furthermore, the APX-hardness reduction to MaxMPM inherently requires non-uniform edge weights and does not translate to MIM. To the best of our knowledge, there are no non-trivial approximation algorithms for any of these problems on more than three stages.

2.3 A NOTE ON APPROXIMATING MAXMPM

For MinMPM on metric spanning 2- or 3-stage graphs, the authors of [BEL+18] show 3-approximations. They also propose a $1/2$-approximation for MaxMPM on spanning multistage graphs with an arbitrary number of stages, which is unfortunately wrong.

It takes a multistage graph as input, where each stage may be an arbitrary graph (not necessarily complete), picks a matching for every second stage G_i, and reuses the same matching for stage G_{i+1}. Thus, every second stage transition is optimal, whereas every other second transition potentially constitutes a worst case. *If* the algorithm's solution is feasible, we indeed obtain the proposed approximation ratio. However, such an approach is inherently problematic as there is no reason why a matching in G_i would need to be feasible for G_{i+1}. In fact, consider a multistage graph $\mathcal{G} = (G_1, \dots, G_\tau)$ where $V_i = \{v_1, \dots, v_4\}$ for all $i \in [\tau]$, $E_i = \{v_1v_2, v_3v_4\}$ for odd i, and $E_i = \{v_2v_3, v_4v_1\}$ for even i. As no perfect matching in E_i is also a perfect matching in E_{i+1}, we have $p(\mathcal{M}) = 0$ for any multistage perfect matching $\mathcal{M}$ for $\mathcal{G}$. Since one could argue about simple methods to devalue this example (see Section 2.5 for an efficient preprocessing rendering this instance trivial), we will provide a second, more involved counterexample.

Thus, although any α-approximation for MaxMPM would directly yield an α-approximation for MIM on spanning multistage graphs, we currently do not know of any such algorithm. In fact, a constant-factor approximation seems difficult to obtain, see Theorem 7. Personal communication with B. Escoffier confirmed our counterexample. One may consider a relaxed version of MaxMPM where one tries to find matchings of large weight in each stage, formally optimizing the weighted sum between the profit and the summed stagewise matching weights. Observe that in this scenario it is not guaranteed that the optimal

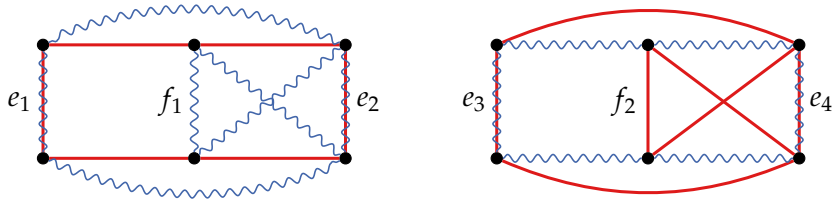


Figure 2.1: Counterexample for the proposed 1/2-approximation. Edges in $E_1{=}E_3$ are curvy (and blue), edges in $E_2{=}E_4$ are straight (and red).

solution induces a perfect (nor even maximum) matching in each stage. However, for this problem their analysis would be correct and their algorithm yields a 1/2-approximation.

COUNTEREXAMPLE. We examine the 1/2-approximation algorithm $\mathcal{A}$ for MAXMPM that was proposed in [BEL+18, Theorem 8], and give a spanning 4-stage instance (that is also *reduced* in the sense of Section 2.5) where $\mathcal{A}$ does not yield a feasible solution. We use four stages since the algorithm treats fewer stages as special cases. Still, the feasibility problem that we are about to describe is inherent to all its variants.

Consider the multistage graph $\mathcal{G} = (G_1, G_2, G_3, G_4)$ given in Figure 2.1, where $E_1 = E_3$ and $E_2 = E_4$ and edges have uniform weight 0. We trivially observe that any perfect matching in G_i is optimal w.r.t. edge weight. For each $i \in [3]$, we have $E^i_\cap := E_i \cap E_{i+1} = \{e_1, e_2, e_3, e_4\}$.

The algorithm proceeds as follows on $\mathcal{G}$: For each $i \in [3]$, it computes a perfect matching M_i in G_i that maximizes $|M_i \cap E_{i+1}|$. It constructs the solutions $\mathcal{M} := (M_1, M_1, M_3, M_3)$ and $\mathcal{M}' := (\hat{M}_1, M_2, M_2, \hat{M}_3)$, where $\hat{M}_i$ is an arbitrary perfect matching in G_i, and outputs the solution that maximizes the profit.

Any perfect matching M_1 in G_1 that maximizes $|M_1 \cap E_2|$ must contain both e_1 and e_2 and as such also f_1. This contradicts the feasibility of $\mathcal{M}$, as $f_1 \notin E_2$. Conversely, any such perfect matching M_2 in G_2 must contain both e_3 and e_4 and as such also f_2. Again, this contradicts the feasibility of $\mathcal{M}'$, since $f_2 \notin E_3$. It follows that the algorithm cannot pick a feasible solution. We are not aware of any way to circumvent this problem.

2.4 CONTRIBUTION

We start with showing in Chapter 3 that the problems at hand are NP-hard even in much more restricted scenarios than previously known, and that (a lower bound for) the integrality gap of the natural linear program for MIM$|_2$ is close to the approximation ratio we will subsequently devise. This hints that stronger approximation ratios may be hard to obtain, at least using LP techniques.

As our main contribution, we propose several approximation algorithms for the multistage problems MIM and MUM, as well as for their stage-restricted variants, see Figure 2.2. The algorithms' approximation ratios are dependent on the instance-specific parameter of intertwinement $\chi := \max_{i \in [\tau-1]} |E_i \cap E_{i+1}|$.

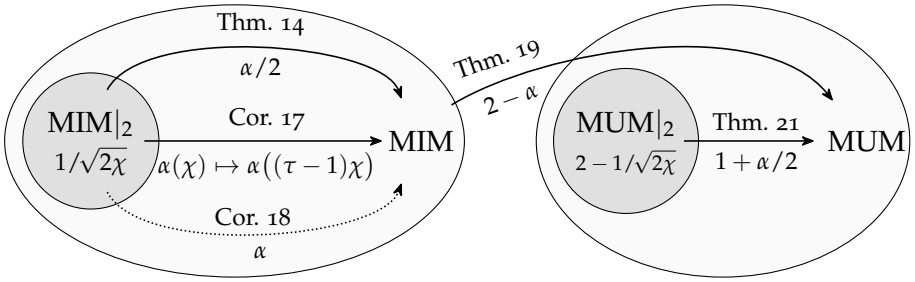


Figure 2.2: Relations of approximation results for multistage matching problems. An arc from problem A to B labeled $f(\alpha)$ denotes the existence of an $f(\alpha)$-approximation for B, given an α-approximation for A. In Cor. 18, α has to be constant. In Cor. 17, $\alpha(\cdot)$ is a function of χ. The ratio of MIM$|_2$ is by Thm. 8; combining this with Thm. 19 yields the ratio for MUM$|_2$.

In particular, in Section 4.1, we present a $1/\sqrt{2\chi}$-approximation for MIM$|_2$ and show that this analysis is tight. Then, in Section 4.2, we show that any approximation of MIM$|_2$ can be used to derive two different approximation algorithms for MIM, whose approximation ratios are a priori incomparable. In Section 4.3, we further show how to use all these algorithms to approximate MUM (and MUM$|_2$). We also observe that it is infeasible to use an arbitrary MUM algorithm to approximate MIM. In particular, we propose the seemingly first approximation algorithms for MIM and MUM on arbitrarily many stages. We stress that our goal is to always guarantee a perfect matching in each stage; the approximation ratio deals purely with optimizing the transition costs. Recall that approximation algorithms optimizing a weighted sum between stage-wise objective and transition quality cannot guarantee such solutions in general.

2.5 PREPROCESSING AND REMARKS

Given a graph $G = (V, E)$, a single edge e is *allowed* if there exists a perfect matching M in G with $e \in M$ and *forbidden* otherwise. A graph is *matching-covered* if all its edges are allowed (see [LP86] for an in-depth characterization of matching-covered graphs). Forbidden edges can easily be found in polynomial time; see e.g. [RV89] for an efficient algorithm. A simple preprocessing for MIM and MUM is to remove all forbidden edges in each stage, as they will never be part of a multistage matching. Thereby, we obtain an equivalent *reduced* multistage graph, i.e., a multistage graph whose stages are matching-covered. If any stage in the reduced multistage graph contains no edges (but vertices), the instance is infeasible. In the following, we thus assume w.l.o.g. that the given multistage graph is reduced and *feasible*, i.e., there exists some perfect matching in each stage.

Remark 3. *Let $\mathcal{G}$ be a reduced 2-stage graph. For any $e \in E_\cap$, there is a perfect matching in each stage that includes e. Thus, there is a multistage perfect matching with profit at least 1 if $E_\cap \neq \varnothing$.*

Remark 4. *For any multistage perfect matching $(M_i)_{i\in[\tau]}$, it holds for each $i \in [\tau - 1]$ that*

$$\max(\frac{n_i}{2}, \frac{n_{i+1}}{2}) \leq |M_i \cup M_{i+1}| = c(M_i, M_{i+1}) \leq 2\max(\frac{n_i}{2}, \frac{n_{i+1}}{2}).$$

Computing any multistage perfect matching is thus an immediate 2-approximation for MUM.

Remark 5. *Consider the following naïve algorithm: Enumerate all sequences $(F_i)_{i\in[\tau-1]}$ where $F_i \subseteq E_i \cap E_{i+1}$ for each $i \in [\tau - 1]$; then check for each $i \in [\tau]$ whether there is a perfect matching M_i in G_i such that $F_{i-1} \cup F_i \subseteq M_i$, where $F_0 = F_\tau = \varnothing$. Thus,* MIM *and* MUM *are in FPT w.r.t. parameter $\sum_{i\in[\tau-1]} |E_i \cap E_{i+1}|$ (or similarly $\tau \cdot \chi$).*

3

SETTING THE GROUND

Before we present our main contribution, the approximation algorithms, we motivate the intrinsic complexities of the considered problems. On the one hand, we show that the problem is already hard in very restricted cases. On the other hand, we show that natural linear programming methods cannot yield a constant-factor approximation for MIM$|_2$.

3.1 NP-HARDNESS

While we have already established that MIM$|_2$ is NP-hard in general, we show that MIM$|_2$ is already NP-hard in the seemingly simple case where each vertex has only degree 2 in both stages. It immediately follows that the decision variants of MIM, MUM$|_2$, MUM, MinMPM, and MaxMPM remain NP-hard as well, even if restricted to this set of multistage graphs.

Theorem 6. *Deciding* MIM$|_2$ *is NP-hard on spanning multistage graphs whose union graph is bipartite, even if both stages consist only of disjoint even cycles and* $E_\cap$ *is a collection of disjoint 2-paths.*

Proof. We will perform a reduction from Maximum Cut [GJ79] to MIM$|_2$. In Maximum Cut, one is given an undirected graph $G = (V, E)$, a natural number k and the question is to decide whether there is an $S \subseteq V$ such that $|\delta(S)| \geq k$. In the first stage, we will construct an even cycle for each vertex and each edge of the original graph and in the second stage we will create an even cycle for each incidence between an edge and a vertex (see Figure 3.1). A perfect matching in the first stage will correspond to a vertex selection and a perfect matching in the second stage will allow us to count the edges that are incident to exactly one selected vertex.

Given an instance $\mathcal{I} := (G = (V, E), k)$ of Maximum Cut, we construct an instance $\mathcal{J} := (\mathcal{G}, \kappa)$ of MIM$|_2$. Set $\kappa := 3|E| + k$. We start with an empty 2-stage graph $\mathcal{G} := (G_1, G_2)$.

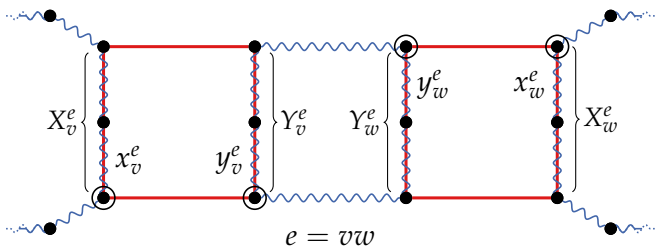


Figure 3.1: Thm. 6: E_1 is curvy blue, E_2 is straight red. Marked vertices are encircled.

Let $I := \{(v,e) \mid v \in V, e \in \delta(v)\}$ be the set of incidences. For each $(v,e) \in I$, we add two new disjoint 2-paths to $E_1 \cap E_2$ and call them X_v^e and Y_v^e. Mark one edge of each X_v^e as x_v^e and one edge of each Y_v^e as y_v^e. We will refer to the endpoint of X_v^e (Y_v^e) incident to x_v^e (y_v^e) as the *marked endpoint* of X_v^e (respectively Y_v^e).

In G_1, for each $e = vw \in E$, we create a 6-cycle through Y_v^e and Y_w^e by adding an edge between the marked endpoint of one path and the unmarked endpoint of the other, and vice versa. Furthermore, for each $v \in V$, we create a cycle of length $4|\delta(v)|$ through the paths X_v^e for $e \in \delta(v)$ as follows: For each edge $e \in \delta(v)$ and its successor edge f, according to some arbitrary cyclic order of $\delta(v)$, connect the marked endpoint of X_v^e to the unmarked endpoint of X_v^f by a 2-path with a new inner vertex.

In G_2, for each $(v,e) \in I$, we generate a 6-cycle through X_v^e and Y_v^e by adding an edge between the marked and an edge between the unmarked endpoints of the 2-paths, respectively. G_1 consists of $|V| + |E|$ and G_2 of $2|E|$ disjoint even cycles, thus $\mathcal{G}$ is reduced.

Claim. *$\mathcal{J}$ is a yes-instance if and only if $\mathcal{I}$ is a yes-instance.*

Proof of Claim. Since both stages of $\mathcal{G}$ consist only of pairwise disjoint even cycles and there are only two perfect matchings in an even cycle, a perfect matching in a stage is determined by choosing one edge in each cycle. For $e = vw \in E$, let $X^e := E(X_v^e) \cup E(X_w^e)$ and $Y^e := E(Y_v^e) \cup E(Y_w^e)$ denote those common edges that correspond to e. Since those are the only common edges between the two stages, we have $E_\cap = \biguplus_{e \in E} X^e \cup Y^e \supseteq M_1 \cap M_2$ for any multistage perfect matching (M_1, M_2).

"$\Leftarrow$" Suppose there is an $S \subseteq V$ such that $|\delta(S)| \geq k$. For each $(v,e) \in I$, add x_v^e to both M_1 and M_2 if $v \in S$. Otherwise, add the unmarked edge of X_v^e to M_1 and M_2. This uniquely determines a perfect matching M_2 in G_2, where $y_v^e \in M_2 \iff x_v^e \in M_2 \iff v \in S$. For each $e = vw \in \delta(S)$ with $v \in S$ and $w \notin S$, add y_v^e to M_1; thus $y_w^e \notin M_1$ and $y_v^e \in M_1 \cap M_2$. For $e = vw \notin \delta(S)$, add either y_v^e or y_w^e to M_1 (chosen arbitrarily). This determines a 2-stage perfect matching (M_1, M_2).

Consider some edge $e = vw \in E$. The intersection $M_1 \cap M_2$ contains two edges of X^e. If $e \in \delta(S)$, it also contains two edges of Y^e, one marked and one unmarked. If $e \notin \delta(S)$, $M_1 \cap M_2$ contains exactly one edge of Y^e. Thus, $|M_1 \cap M_2| = |\biguplus_{e \in E} M_1 \cap M_2 \cap X^e| + |\biguplus_{e \in E} M_1 \cap M_2 \cap Y^e| = 3|E| + |\delta(S)|$.

"$\Rightarrow$" Let (M_1, M_2) be a multistage perfect matching in $\mathcal{G}$ with large intersection $|M_1 \cap M_2| \geq 3|E| + k$. For $e \in E$, let $m_e := |M_1 \cap M_2 \cap (X^e \cup Y^e)| \leq 4$ for each $e \in E$. Observe that $|M_1 \cap M_2| = \sum_{e \in E} m_e$. Thus, by pigeonhole principle, there are at least k edges with $m_e = 4$.

Now M_1 yields a selection $S \subseteq V$: Select $v \in V$ if and only if the set $X(v) := \{x_v^e \mid e \in \delta(v)\}$ is contained in M_1. In a perfect matching in G_1, either all or none of the edges in $X(v)$ are matched simultaneously. It can be seen that $m_e = 4$ if and only if $e \in \delta(S)$, thus $|\delta(S)| \geq k$. ◁

The cycles of length $4|\delta(v)|$ may have introduced an even number of vertices $W \subseteq V$ that are isolated in G_2. To make $\mathcal{G}$ spanning, we may add to E_2 an even cycle on W. This neither interferes with $E_\cap$ nor the profit p, since W is an independent set in the first stage G_1. □

3.2 LINEAR PROGRAMMING GAP

Linear programs (LPs)—as relaxations of integer linear programs (ILPs)—are often used to provide dual bounds in the approximation context [LP86; WS11; Vaz03]. Here, we consider the natural LP-formulation of MIM$|_2$ and show that the integrality gap (i.e., the ratio between the optimal objective value of the ILP and the optimal objective value of its relaxation) is at least $\sqrt{\chi}$, already for spanning instances with a bipartite union graph. Up to a small constant factor, this equals the (inverse) approximation ratio guaranteed by Algorithm 9, which we will propose in Chapter 4. This serves as a hint that overcoming the approximation dependency on $\sqrt{\chi}$ for MIM$|_2$ may be hard.

In the context of classical (perfect) matchings, the standard ILP-formulation and its LP-relaxation describe the very same feasible points, the so-called *matching polytope*, which is the corner stone of the problem being solvable in polynomial time as described in the seminal paper by Edmonds [Edm65a]. Given a 2-stage graph $\mathcal{G} = (G_1, G_2)$, the natural LP-formulation for MIM$|_2$ starts with the product of two distinct such perfect matching polytopes. Let $\delta_\ell(v)$ denote all edges incident to vertex v in G_ℓ, and let (M_1, M_2) be a 2-stage perfect matching in $\mathcal{G}$. For each $\ell \in [2]$, we model M_ℓ via the standard matching polytope: For each $e \in E_\ell$ there is an indicator variable x_e^ℓ that is 1 if and only if $e \in M_\ell$. The constraints (3.1a) below suffice for bipartite graphs; for general graphs one also considers the *blossom constraints* (3.1b). These ensure that for any odd-sized vertex set W, at most $\lfloor |W|/2 \rfloor$ edges between those vertices can partake in a matching (see [Edm65a; LP86] for details). Additionally to these standard descriptions, for each $e \in E_\cap$ we use a variable z_e that is 1 if and only if $e \in M_1 \cap M_2$. This yields the following ILP:

$$\max \sum_{e \in E_\cap} z_e$$

$$\begin{aligned}
s.t. \quad \sum_{e \in \delta_\ell(v)} x_e^\ell &= 1 && \forall\, \ell \in [2],\ \forall\, v \in V_\ell && \text{(3.1a)}\\
\sum_{e \in E(G[W])} x_e^\ell &\leq \frac{|W|-1}{2} && \forall\, \ell \in [2],\ \forall\, W \subseteq V_\ell \text{ with odd } |W| && \text{(3.1b)}\\
z_e &\leq x_e^\ell && \forall\, \ell \in [2],\ \forall\, e \in E_\cap && \text{(3.1c)}\\
x_e^\ell &\in \{0,1\} && \forall\, \ell \in [2],\ \forall\, e \in E_\ell && \text{(3.1d)}\\
z_e &\in \{0,1\} && \forall\, e \in E_\cap && \text{(3.1e)}
\end{aligned}$$

Thereby, constraints (3.1c), together with the fact that we maximize all z-values, ensure that $z_e = \min\{x_e^1, x_e^2\}$ in any optimal solution.

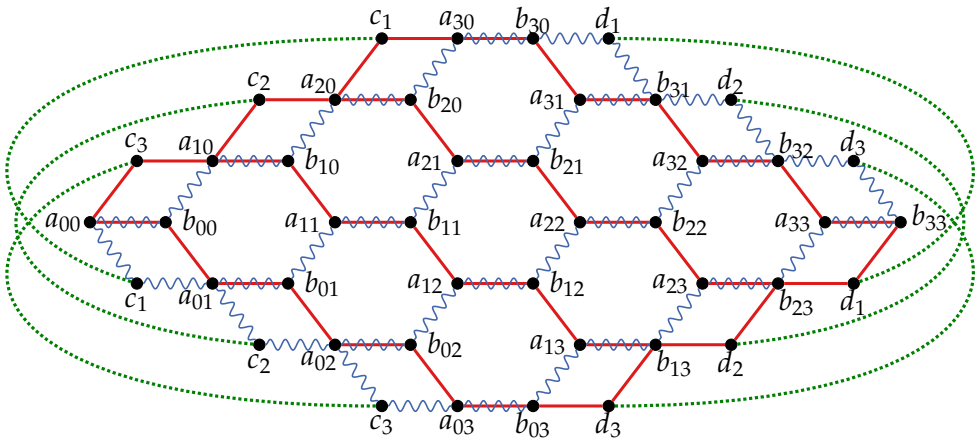


Figure 3.2: MIM$|_2$ instance for k=3 with an integrality gap $\geq\sqrt{\chi}$. E_1 is straight and red, E_2 is curvy and blue. Dotted green lines identify vertices.

Theorem 7. *The natural LP for* MIM$|_2$ *has at least an integrality gap of* $\sqrt{\chi}$.

Proof. We construct a family of MIM$|_2$ instances, each with bipartite union graph, parameterized by a parameter k. Each instance is reduced and has maximum profit of 1, but its LP-relaxation has objective value at least $k+1=\sqrt{\chi}$.

Fix some $k \geq 3$. We construct $\mathcal{G} := \mathcal{G}(k) = (G_1, G_2)$ as follows (see Figure 3.2 for a visualization with $k=3$). Let $V(\mathcal{G}) := \{a_{i,j}, b_{i,j}\}_{i,j\in[\![k]\!]} \cup \{c_i, d_i\}_{i\in[k]}$. Let $E_\cap := \{a_{i,j}b_{i,j}\}_{i,j\in[\![k]\!]}$, i.e., the intersection contains precisely the natural pairings of the a and b vertices. We call these the *shared* edges. In E_1, we additionally add edges $\{b_{i-1,j}a_{i,j}\}_{i\in[k],j\in[\![k]\!]}$. Similarly, we add edges $\{b_{i,j-1}a_{i,j}\}_{i\in[\![k]\!],j\in[k]}$ to E_2. Now, both stages consist of $k+1$ disjoint paths of $2k+1$ edges which are "interwoven" between the stages such that (i) every second edge in each path is shared (starting with the first), and (ii) any path in G_1 has exactly one edge in common with every path in G_2. Let P_i^ℓ, $\ell \in [2], i \in [\![k]\!]$, denote those paths in their natural indexing. We make each stage connected by joining every pair of "neighboring" paths, each together with a c and a d vertex. More precisely, we add edges $\{c_j a_{0,j-1}, c_j a_{0,j}, b_{k,j-1}d_j, b_{k,j}d_j\}_{j\in[k]}$ to E_1. Analogously, we add $\{c_{\varphi(i)}a_{i-1,0}, c_{\varphi(i)}a_{i,0}, b_{i-1,k}d_{\varphi(i)}, b_{i,k}d_{\varphi(i)}\}_{i\in[k]}$ to E_2; the indexing function $\varphi(i) := k-i+1$ ensures that these new edges are not common to both stages. (In fact, if we would not care for a spanning $\mathcal{G}$, we could simply use "new" vertices instead of reusing c, d in G_2.) This finishes the construction, and since $\mathcal{G}$ contains no forbidden edges, it is reduced.

Since the inner vertices of any path P_i^1 have degree 2 in G_1, any perfect matching in G_1 either contains all or none of the path's shared edges. Assume some shared edge $a_{0,j}b_{0,j}$ is in a perfect matching in G_1. Let C be the path $a_{0,0}\, c_1\, a_{0,1}\, c_2\, \ldots\, c_k\, a_{0,k}$ in G_1. Recall that all c-vertices have degree 2 in G_1. Since $a_{0,j}$ is matched outside of C, all other $a_{0,j'}$ with $j' \neq j$ have to be matched with these c-vertices. Thus, P_j^1 is the only path that contributes shared edges to the matching. Conversely, since C contains one less c-vertex than a-vertices, any perfect matching in G_1 has to have at least (and thus exactly) one such path.

As the analogous statement holds for G_2 and by the interweaving property (ii) above, any multistage perfect matching contains exactly one shared edge.

However, we construct a feasible fractional solution with objective value $\sqrt{\chi}$. Let $\lambda := 1/(k+1)$. We set the x- and z-variables of all shared edges to λ, satisfying all constraints (3.1c). This uniquely determines all other variable assignments, in order to satisfy (3.1a): Since the inner vertices of each P_i^ℓ have degree 2 in G_ℓ, the non-shared edges in these path have to be set to $1-\lambda$. Again consider path C: Each a-vertex in C has an incident shared edge that contributes λ to the sum in the vertex' constraint (3.1a); there are no other edges incident to C. Thus, we have to set $x^1_{c_j a_{0,j-1}} = 1 - j\lambda$ and $x^1_{c_j a_{0,j}} = j\lambda$ such that, for each vertex in C, its incident variable values sum to 1. The analogous statements holds for the corresponding path through d-vertices in G_1, and the analogous paths in G_2. All constraints (3.1a) are satisfied. The blossom constraints (3.1b) act only on x-variables, i.e., on individual stages. Since our graph is bipartite, only considering the x-variables of one stage and disregarding (3.1b) yields the bipartite matching polytope which has only integral vertices; our (sub)solution is an element of this polytope. Thus, constraints (3.1b) cannot be violated by our assignment.

By construction we have $\chi = (k+1)^2$. Thus, the objective value of our assignment is $\sum_{e \in E_\cap} \lambda = \chi/(k+1) = \sqrt{\chi}$, as desired. □

4

APPROXIMATION

We start with the special case of MIM$|_2$, before extending the result to the multistage MIM scenario. Then we will transform the algorithms for use with MUM$|_2$ and MUM.

4.1 APPROXIMATING MIM$|_2$

We first describe Algorithm 9, which is an approximation for MIM$|_2$. Although its ratio is not constant but grows with the rate of $\sqrt{\chi}$, Theorem 7 hints that better approximations may be hard to obtain. Algorithm 9 roughly works as follows: Given a 2-stage graph $\mathcal{G}$, we iterate the following procedure on G_1 until every edge of $E_\cap$ has been in at least one perfect matching: Compute a perfect matching M_1 in G_1 that uses the maximum number of edges of $E_\cap$ that have not been used in any previous iteration; then compute a perfect matching M_2 in G_2 that optimizes the profit with respect to M_1. While doing so, keep track of the maximal occurring profit. Note that by choosing weights appropriately, we can construct a perfect matching that contains the maximum number of edges of some prescribed edge set in polynomial time [LP86]. We show:

Theorem 8. *Algorithm 9 is a tight $1/\sqrt{2\chi}$-approximation for* MIM$|_2$.

We prove this via two lemmata; the bad instance of Lemma 10 in conjunction with the approximation ratio (Lemma 11) establishes tightness.

Lemma 10 (Bad instance)**.** *The approximation ratio of Algorithm 9 is at most $1/\sqrt{2\chi}$.*

Proof. Consider the following family $\mathcal{G}_k$ of MIM$|_2$ instances, parameterized by some $k \geq 1$. An example using $k = 4$ is depicted in Figure 4.1. In the first stage, for each $i \in [k]$ create a 4-cycle C_i and label two of its adjacent vertices w'_i and w_i. Add a 3-path with new inner vertices of degree 2 from w_i to w'_{i+1} for each $i \in [k-1]$. For each $i \in [k-1]$, create a vertex v_i and an edge $w_i v_i$. Create a vertex u and an edge uw_k. For each $i \in [\![k-1]\!]$, create a vertex u_i and an edge uu_i. For each $i \in [k-1]$, create a path P_i from u_i to v_i with $2i+1$ edges and label the new inner vertices with a^i_1, b^i_1, a^i_2, b^i_2, …, a^i_i, b^i_i in this order.

The second stage is constructed isomorphically to the first stage. To avoid ambiguity in the naming, we underline element names of the second stage. The 2-stage graph $\mathcal{G}_k$ is completely defined by the following identifications: for each $i \in [k]$, let $\underline{w}'_i = w_{k-i+1}$ and $\underline{w}_i = w'_{k-i+1}$; for each $i \in [k-1]$ and each $j \in [i]$, let $\underline{a}^i_j = b^{k-j}_{k-i}$ and $\underline{b}^i_j = a^{k-j}_{k-i}$. Thus, $E_\cap = F \cup A$ is precisely the union of $F := \{w_i w'_i \mid i \in [k]\}$ and $A := \{a^i_j b^i_j \mid i \in [k-1], j \in [i]\}$. Observe that $\mathcal{G}_k$ is reduced, its union graph is bipartite, and $\chi = k + \sum_{i \in [k-1]} i = k(k+1)/2$.

Algorithm 9: Approximation of MIM$|_2$

Input: Weighted 2-stage graph $\mathcal{G} = (G_1, G_2)$
Output: 2-stage perfect matching (M_1, M_2)

$(M_1, M_2) \leftarrow (\varnothing, \varnothing)$
for $i = 1, 2, \ldots$ **do**
  $w_1(e) \leftarrow \mathbb{1}\big(e \in E_\cap \setminus \bigcup_{j\in[i-1]} M_1^{(j)}\big)$ for $e \in E_1$
  compute a maximum weight perfect matching $M_1^{(i)}$ in G_1 w.r.t. w_1
  $w_2(e) \leftarrow \mathbb{1}\big(e \in M_1^{(i)}\big)$ for $e \in E_2$
  compute a maximum weight perfect matching $M_2^{(i)}$ in G_2 w.r.t. w_2
  if $|M_1^{(i)} \cap M_2^{(i)}| \geq |M_1 \cap M_2|$ **then** $(M_1, M_2) \leftarrow (M_1^{(i)}, M_2^{(i)})$
  if $E_\cap \subseteq \bigcup_{j\in[i]} M_1^{(j)}$ **then return** (M_1, M_2)

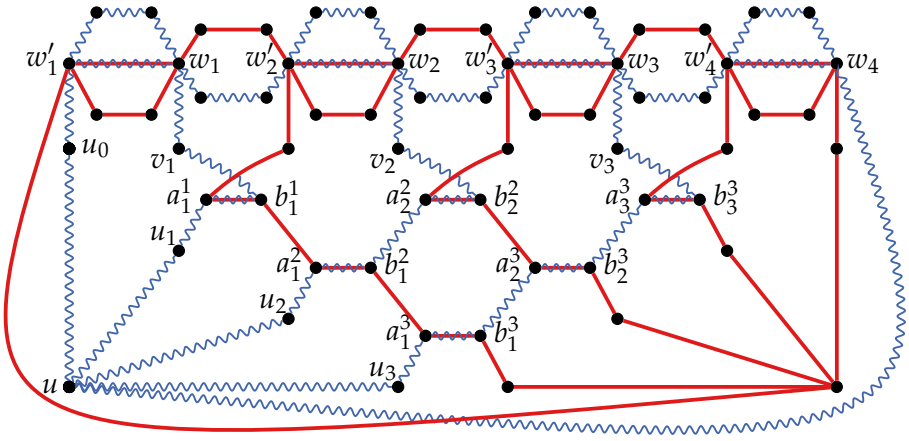


Figure 4.1: MIM$|_2$ instance $\mathcal{G}_4$ as in Lemma 10. Edges in E_1 are curvy and blue, edges in E_2 straight and red. The vertices are labeled according to the first stage.

By construction, for any perfect matching M in the first (or second) stage, we have $|M \cap E_\cap| \leq k$. Let M_F ($\underline{M}_F$) denote the unique perfect matching that contains uu_0 ($\underline{u}\,\underline{u}_0$, respectively) and all of F. The pair $(M_F, \underline{M}_F)$ is an optimal solution with profit $|F| = k$.

Consider an alternative perfect matching M in the first stage. In each cycle C_i, we consider the shared edge $w_i w'_i$ and its *opposing edge* (i.e., its unique non-adjacent edge in C_i). We distinguish between three possibilities regarding their memberships in M: the shared edge and its opposing edge are in M (*type Y*), only the opposing edge is in M (*type N1*), none of them are in M (*type N2*).

Picking an edge adjacent to u determines a perfect matching up to the types of some C_i-cycles. For $i \in [\![k-1]\!]$, let M_i denote the unique perfect matching that contains uu_i, is type Y in C_{i+1}, but type $N2$ in each C_j for $j > i+1$. Note that M_i is type $N1$ in each C_j with $j \leq i$ and contains all i shared edges along P_i. Thus, $|M_i \cap E_\cap| = i + 1$. Most importantly, consider any perfect matching M' in the

second stage. By construction, for any $i \in [\![k-1]\!]$, no two edges of $(M_i \cap A) \cup \{w'_{i+1}w_{i+1}\}$ can be contained simultaneously in M'. It follows that $|M_i \cap M'| \leq 1$.

Algorithm 9 may never choose the optimal M_F as a perfect matching for the first stage: In the first iteration, both M_F and M_{k-1} have weight k, so the algorithm may choose M_{k-1} and obtain a 2-stage perfect matching with profit 1. In the following iteration, the weight (denoting the preference of edges) of M_F is decreased by 1, since the edge $w'_k w_k$ has already been chosen in M_{k-1}. Consequently, in each following iteration $i \in [k]$ the algorithm may choose M_{k-i} over M_F, each time decreasing the weight of M_F by 1. After choosing M_0 over M_F in iteration k, each edge in $E_\cap$ has been in some matching in the first stage; the algorithm stops and returns a 2-stage perfect matching with profit 1.

Since $\chi = k(k+1)/2$, the optimal profit is $k = \left(\sqrt{8\chi+1}-1\right)/2$. Thus, the approximation factor is at most $1/k = 2/\left(\sqrt{8\chi+1}-1\right)$ which tends to our approximation ratio of $1/\sqrt{2\chi}$ for increasing k. □

Lemma 11 (Guarantee)**.** *The approximation ratio of Algorithm 9 is at least* $1/\sqrt{2\chi}$*.*

Proof. Let $\mathcal{G}$ be a feasible and reduced 2-stage graph with non-empty $E_\cap$. Clearly, our algorithm achieves $\mathsf{apx} \geq 1$ as described in Remark 3. In each iteration i of the loop, the algorithm picks into $M_1^{(i)}$ at least one edge of $E_\cap$ that has not been in any previous $M_1^{(j)}$ (otherwise the loop terminates) and hence, the loop terminates in polynomial time. Let k denote the number of iterations. For any $i \in [k]$, let $(M_1^{(i)}, M_2^{(i)})$ denote the 2-stage perfect matching computed in the ith iteration. Let (M_1^*, M_2^*) denote an optimal 2-stage perfect matching and $M_\cap^* := M_1^* \cap M_2^* \subseteq E_\cap$ its intersection (note that $M_1^* \cap E_\cap \setminus M_\cap^*$ may be non-empty). Let $R_i := (M_1^{(i)} \cap E_\cap) \setminus \bigcup_{j\in[i-1]} R_j$ denote the set of intersection edges that are in $M_1^{(i)}$ but not in $M_1^{(j)}$ for any previous iteration $j < i$; let $r_i := |R_i|$. Note that in iteration i, the algorithm first searches for a perfect matching $M_1^{(i)}$ in G_1 that maximizes

$$|M_1^{(i)} \cap (E_\cap \setminus \bigcup_{j\in[i-1]} M_1^{(j)})| = |M_1^{(i)} \cap (E_\cap \setminus \bigcup_{j\in[i-1]} R_j)| = r_i.$$

We define $R_i^* := (M_1^{(i)} \cap M_\cap^*) \setminus \bigcup_{j\in[i-1]} R_j^*$ and $r_i^* := |R_i^*|$ equivalently to R_i, but w.r.t. $M_\cap^*$ instead of $E_\cap$ (see Figure 4.2). Thus, R_i^* contains those edges of $M_\cap^*$ that are selected (into $M_1^{(i)}$) for the first time over all iterations. Observe that $R_i \cap M_\cap^* = R_i^*$.

Let $x := \sqrt{2\chi}$. For every $i \in [k]$, the algorithm chooses a perfect matching $M_2^{(i)}$ in G_2 that maximizes $|M_1^{(i)} \cap M_2^{(i)}|$. Since we may choose $M_2^{(i)} = M_2^*$, it follows that $\mathsf{apx} \geq \max_{i\in[k]} r_i^*$. Thus, if $\max_{i\in[k]} r_i^* \geq \mathsf{opt}/x$, we have a $1/x$-approximation. In case $\mathsf{opt} \leq x$, any solution with profit at least 1 (cf. Remark 3) yields a $1/x$-approximation. We show that we are always in one of the two cases.

Assume that $\mathsf{opt} > x$ and simultaneously $r_i^* < \mathsf{opt}/x$ for all $i \in [k]$. Since we distribute $M_\cap^*$ over the disjoint sets $\{R_i^* \mid i \in [k]\}$, each containing less than opt/x edges, we know that $k > x$ (thus $k \geq \lceil x \rceil =: \bar{x}$). Recall that in iteration i, we have

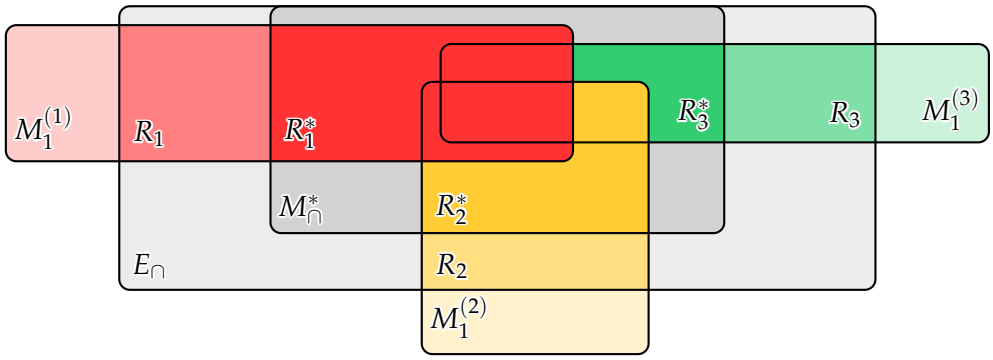


Figure 4.2: Inclusion relationships between $E_\cap$, $M_\cap^*$, $M_1^{(i)}$, R_i, and R_i^* for $i \in [3]$.

that $(E_\cap \setminus \bigcup_{j\in[i-1]} M_1^{(j)}) \cap \bigcup_{j\in[i-1]} R_j$ is empty. Thus, the number of elements of M_1^* that are counted towards $|M_1^{(i)} \cap (E_\cap \setminus \bigcup_{j\in[i-1]} M_1^{(j)})| = r_i$ is

$$\big|(M_1^* \cap E_\cap) \setminus \bigcup_{j\in[i-1]} R_j\big| \geq \big|M_\cap^* \setminus \bigcup_{j\in[i-1]} R_j\big| = \big|M_\cap^* \setminus \bigcup_{j\in[i-1]} R_j^*\big|.$$

Hence, the latter term is a lower bound on r_i and we deduce:

$$\begin{aligned} r_i &\geq \big|M_\cap^* \setminus \bigcup_{j\in[i-1]} R_j^*\big| = \mathsf{opt} - \sum_{j\in[i-1]} r_j^* \\ &\overset{(\star)}{\geq} \mathsf{opt} - \sum_{j\in[i-1]} \frac{\mathsf{opt}}{x} = \mathsf{opt} \cdot \big(1 - \frac{i-1}{x}\big) \geq \bar{x}\big(1 - \frac{i-1}{x}\big). \end{aligned}$$

Thereby, strict inequality holds at $(\star)$ for $i \geq 2$. This raises a contradiction:

$$\begin{aligned} \chi = |X_\cap| &= \big|\bigcup_{i\in[k]} R_i\big| \geq \sum_{i\in[\bar{x}]} r_i \gneq \sum_{i\in[\bar{x}]} \bar{x}\big(1 - \frac{i-1}{x}\big) \\ &= \bar{x} \cdot \Big(\sum_{i\in[\bar{x}]} 1 - \sum_{i\in[\bar{x}-1]} \frac{i}{x}\Big) = \bar{x}\big(\bar{x} - \frac{(\bar{x}-1)\bar{x}}{2x}\big) \\ &= \bar{x}^2\big(1 - \frac{\bar{x}-1}{2x}\big) \geq \bar{x}^2\big(1 - \frac{x}{2x}\big) = \frac{\bar{x}^2}{2} \geq \frac{x^2}{2} = \chi. \end{aligned}$$

□

4.2 APPROXIMATING MIM

Let us extend the above result to an arbitrary number of stages. We show that we can use *any* MIM$|_2$ approximation algorithm (in particular also Algorithm 9) as a black box to obtain an approximation algorithm for MIM, while only halving the approximation ratio: Algorithm 13 uses an edge-weighted path (P, w) on τ vertices as an auxiliary graph. We set the weight of the edge between the ith and $(i+1)$th vertex to an approximate solution for the MIM$|_2$ instance that arises from the ith and $(i+1)$th stage of the MIM instance. A maximum weight matching M_P in (P, w) induces a feasible solution for the MIM problem: If an edge $(j, j+1)$ is in M_P, we use the corresponding solutions for the jth

Algorithm 13: General multistage approximation

Input: Multistage graph $\mathcal{G}$, 2-stage perfect matching algorithm $\mathcal{A}$
Output: Multistage maximum weight perfect matching $(M_1, \dots, M_\tau)$

create path $P := \{e_1, \dots, e_{\tau-1}\}$
foreach $i \in [\tau - 1]$ **do**
  set (S_i, T_{i+1}) to $\mathcal{A}(G_i, G_{i+1})$ `// approximate 2-stage graphs`
  set weight of e_i to $w_i := |S_i \cap T_{i+1}|$
compute maximum weight matching M_P in (P, w)
set $(M_i)_{i\in[\tau-1]}$ to $(S_i)_{i\in[\tau-1]}$ and M_τ to T_τ `// set initial solution`
foreach $i \in [\tau - 1]$ **do** `// modify solution according to` M_P
  if $e_i \in M_P$ **then** set M_i to S_i and M_{i+1} to T_{i+1}
return $(M_1, \dots, M_\tau)$

and $(j+1)$th stage; for stages without incident edge in M_P, we select an arbitrary solution. Since no vertex is incident to more than one edge in M_P, there are no conflicts.

Remark 12. *For $F \subseteq E(P)$, denote $w(F) := \sum_{e\in F} w(e)$. Let e_i denote the ith edge of P. For $b \in [2]$, the matchings $M_b := \{e_i \in E(P) \mid i = b \mod 2\}$ are disjoint and their union is exactly $E(P)$. Thus, for any maximum weight matching M_P in P we have $2 \cdot w(M_P) \geq w(E(P))$.*

Theorem 14. *For a* MIM$|_2$ *α-approximation, Algorithm 13 $\alpha/2$-approximates* MIM.

Proof. Let $\mathcal{G} = (G_1, \dots, G_\tau)$ be the given multistage graph. For any $i \in [\tau - 1]$, (S_i, T_{i+1}) is the output of the MIM$|_2$ α-approximation $\mathcal{A}(G_i, G_{i+1})$; let $w_i := |S_i \cap T_{i+1}|$. Let $\mathcal{M}^* := (M_1^*, \dots, M_\tau^*)$ denote a multistage perfect matching whose profit $p(\mathcal{M}^*)$ is maximum. Since $\mathcal{A}$ is an α-approximation for MIM$|_2$, we know that $|M_i^* \cap M_{i+1}^*| \leq w_i/\alpha$ for every $i \in [\tau - 1]$. Thus $p(\mathcal{M}^*) \leq (1/\alpha) \sum_{i\in[\tau-1]} w_i$. Algorithm 13 computes a maximum weight matching M_P in (P, w) and constructs a multistage solution $\mathcal{M}$. By Remark 12, we obtain

$$p(\mathcal{M}^*) \leq \frac{1}{\alpha} \sum_{i\in[\tau-1]} w_i = \frac{1}{\alpha} w\big(E(P)\big) \leq \frac{2}{\alpha} w(M_P) \leq \frac{2}{\alpha} p(\mathcal{M}). \qquad \square$$

We compute a maximum weight matching in a path in linear time using straightforward dynamic programming. Hence, assuming a running time function f for $\mathcal{A}$, Algorithm 13 requires $\mathcal{O}\big(\sum_{i\in[\tau-1]} |f(G_i, G_{i+1})|\big)$ steps.

Corollary 15. *Algorithm 9 in Algorithm 13 yields a $1/\sqrt{8\chi}$-approximation for* MIM.

There is another way to approximate MIM via an approximation for MIM$|_2$, which neither dominates nor is dominated by the above method, but is advantageous for small values of τ:

Theorem 16. *There is an S-reduction from* MIM *to* MIM$|_2$*, i.e., given any* MIM *instance* $\mathcal{G}$*, we can find a corresponding* MIM$|_2$ *instance* $\mathcal{G}'$ *in polynomial time such that any solution for* $\mathcal{G}$ *bijectively corresponds to a solution for* $\mathcal{G}'$ *with the same profit. Furthermore,* $|E(G'_1) \cap E(G'_2)| = \sum_{i\in[\tau-1]} |E(G_i) \cap E(G_{i+1})|$.

Proof. We will construct a 2-stage graph $\mathcal{G}'$ whose first stage G'_1 consists of (subdivided) disjoint copies of G_i for odd i; conversely its second stage G'_2 consists of (subdivided) disjoint copies of G_i for even i. More precisely, consider the following construction: Let $b(i) := 2 - (i \bmod 2)$. For each $i \in [\tau]$, we create a copy of G_i in $G'_{b(i)}$ where each edge $e \in E(G_i)$ is replaced by a 7-path p_i^e. We label the 3rd (5th) edge along p_i^e (disregarding its orientation) with e_i^- (e_i^+, respectively). To finally obtain $\mathcal{G}'$, for each $i \in [\tau - 1]$ and $e \in E(G_i) \cap E(G_{i+1})$, we identify the vertices of e_i^+ with those of e_{i+1}^- (disregarding the edges' orientations); thereby precisely the edges e_i^+ and e_{i+1}^- become an edge common to both stages. No other edges are shared between both stages. This completes the construction of $\mathcal{G}'$ and we have $|E(G'_1) \cap E(G'_2)| = \sum_{i\in[\tau-1]} |E(G_i) \cap E(G_{i+1})|$.

Assume $\mathcal{M}' := (M'_1, M'_2)$ is a solution for $\mathcal{G}'$. Clearly, each path p_i^e in $G'_{b(i)}$ is matched alternatingly and hence either all or none of e_i^-, e_i^+, the first, and the last edge of p_i^e are in $M'_{b(i)}$. We derive a corresponding solution $\mathcal{M}$ for $\mathcal{G}$: For every $i \in [\tau]$ and $e \in E(G_i)$, we add e to M_i if and only if $e_i^- \in M'_{b(i)}$. Suppose that M_i is not a perfect matching for G_i, i.e., there exists a vertex v in G_i that is not incident to exactly one edge in M_i. Then also the copy of v in the copy of G_i in $G'_{b(i)}$ is not incident to exactly one edge of $M'_{b(i)}$, contradicting the feasibility of $\mathcal{M}'$.

Consider the profit achieved by $\mathcal{M}$: Every edge in $M'_1 \cap M'_2$ corresponds to a different identification $\langle e_i^+, e_{i+1}^-\rangle$. We have $e \in M_i \cap M_{i+1}$ if and only if $e_i^- \in M'_{b(i)}$, $e_{i+1}^- \in M'_{b(i+1)}$, and $e_i^+ = e_{i+1}^-$. It follows that this holds if and only if $e_i^+ \in M'_{b(i)} \cap M'_{b(i+1)}$ and hence the profit of $\mathcal{M}$ is equal to that of $\mathcal{M}'$. The inverse direction proceeds in the same manner. □

Since the new $\chi' := |E(G'_1) \cap E(G'_2)|$ is largest w.r.t. the original χ if the intersection $|E(G_i) \cap E(G_{i+1})|$ has constant size for all i, we obtain:

Corollary 17. *For any* MIM$|_2$ $\alpha(\chi)$*-approximation, where* $\alpha(\chi)$ *is a (typically decreasing) function of* χ*, there is an* $\alpha\big((\tau-1)\chi\big)$*-approximation for* MIM*. Using Algorithm 9, this yields a ratio of* $1/\sqrt{2(\tau-1)\chi}$*; for* MIM$|_3$ *and* MIM$|_4$ *this is tighter than the approximation in Theorem 14.*

Assume the approximation ratio for MIM$|_2$ would not depend on χ. Then the above would yield a surprisingly strong result:

Corollary 18. *Any* MIM$|_2$ *α-approximation with constant α results in an α-approximation of* MIM*. If* MIM *is APX-hard, so is* MIM$|_2$*.*

4.3 APPROXIMATING MUM

Consider the MUM-problem which minimizes the cost. As noted in Remark 4, a 2-approximation is easily accomplished. However, by exploiting the previous results for MIM, we obtain better approximations.

Theorem 19. *Any α-approximation of* MIM *is a $(2-\alpha)$-approximation of* MUM.

Proof. Recall that an optimal solution of MIM constitutes an optimal solution of MUM. As before, we denote the heuristic sequence of matchings by $(M_i)_{i\in[\tau]}$ and the optimal one by $(M_i^*)_{i\in[\tau]}$. Let $\xi := \sum_{i\in[\tau-1]}(n_i+n_{i+1})/2$. Consider the solutions' values w.r.t. MUM:

$$\frac{\mathsf{apx}_\cup}{\mathsf{opt}_\cup} = \frac{\sum_{i\in[\tau-1]} c(M_i, M_{i+1})}{\sum_{i\in[\tau-1]} c(M_i^*, M_{i+1}^*)} = \frac{\xi - \sum_{i\in[\tau-1]} |M_i \cap M_{i+1}|}{\xi - \sum_{i\in[\tau-1]} |M_i^* \cap M_{i+1}^*|} \leq \frac{\xi - \alpha\cdot\mathsf{opt}_\cap}{\xi - \mathsf{opt}_\cap} =: f.$$

As $0<\alpha<1$, f is monotonously increasing in $\mathsf{opt}_\cap$ if $0 \leq \mathsf{opt}_\cap < \xi$. Thus, since

$$\mathsf{opt}_\cap \leq \sum_{i\in[\tau-1]} \frac{\min(n_i, n_{i+1})}{2} \leq \sum_{i\in[\tau-1]} \frac{n_i+n_{i+1}}{4} = \frac{\xi}{2},$$

it follows that

$$\frac{\mathsf{apx}_\cup}{\mathsf{opt}_\cup} \leq \frac{\xi - \alpha\frac{\xi}{2}}{\xi - \frac{\xi}{2}} = \frac{1-\frac{\alpha}{2}}{1-\frac{1}{2}} = 2-\alpha. \qquad \square$$

Corollary 20. *Let $r := \min\{8, 2(\tau-1)\}$. We have a $\left(2-1/\sqrt{r\cdot\chi}\right)$-approximation for* MUM.

Note that a similar reduction from MIM to MUM is not achieved as easily: Consider any $(1+\varepsilon)$-approximation for MUM. Choose an even $k \geq 6$ such that $k/(k-1) \leq 1+\varepsilon$; consider a spanning 2-stage instance where each stage is a k-cycle and $E_\cap$ consists of a single edge e. The optimal 2-stage perfect matching $\mathcal{M}^*$ that contains e in both stages has profit $p(\mathcal{M}^*) = 1$ and cost $c(\mathcal{M}^*) = 2\cdot k/2 - 1 = k-1$. A 2-stage perfect matching $\mathcal{M}$ that does not contain e still satisfies $c(\mathcal{M}) = k$ and as such is an $(1+\varepsilon)$-approximation for MUM. However, its profit $p(\mathcal{M}) = 0$ does not provide any approximation of $p(\mathcal{M}^*) = 1$.

As for MIM, we aim to extend a given approximation for MUM$|_2$ to a general approximation for MUM. Unfortunately, we cannot use Theorems 16 and 19 for this, as an approximation for MUM$|_2$ does not generally constitute one for MIM$|_2$ (and MIM). On the positive side, a similar approach as used in the proof of Theorem 14 also works for minimization.

Theorem 21. *Any α-approximation $\mathcal{A}$ for* MUM$|_2$ *results in a $(1+\alpha/2)$-approximation for* MUM *by using $\mathcal{A}$ in Algorithm 13.*

Proof. As before, let $(M_i^*)_{i\in[\tau]}$ denote an optimal solution for MUM. For each $i \in [\tau-1]$, let (S_i, T_i) denote the output of $\mathcal{A}(G_i, G_{i+1})$. For $L \subseteq [\tau-1]$, let $\xi(L) :=$

$\sum_{i\in L}(n_i + n_{i+1})/2$ and $\sigma(L) := \sum_{i\in L} |S_i \cup T_i|$. Note that $w_i := \xi(\{i\}) - \sigma(\{i\})$ equals the weight of e_i. We define $I := \{i \in [\tau - 1] \mid e_i \in M_P\}$ as the set of indices corresponding to M_P and $J := [\tau - 1] \setminus I$ as its complement. By Remark 12, we have $w\big(E(P)\big) \leq 2 \cdot w(M_P)$; thus

$$\begin{aligned} &\xi(I) - \sigma(I) + \xi(J) - \sigma(J) = w\big(E(P)\big) \leq 2 \cdot w(M_P) = 2\big(\xi(I) - \sigma(I)\big) \\ \Rightarrow\ &\sigma(I) + \xi(J) \leq \xi(I) + \sigma(J) \Rightarrow 2\big(\sigma(I) + \xi(J)\big) \leq \xi(I \cup J) + \sigma(I \cup J). \end{aligned}$$

The trivial upper bound ξ suffices to bound the algorithm's solution value:

$$\mathsf{apx} = \sigma(I) + \sum_{j\in J} |M_j \cup M_{j+1}| \leq \sigma(I) + \xi(J) \leq \frac{1}{2}\big(\xi(I \cup J) + \sigma(I \cup J)\big).$$

Since $\sigma(I \cup J)$ α-approximates the sum of all MUM$|_2$ instances' solution values, we have $\sigma(I \cup J) \leq \alpha \cdot \mathsf{opt}$. For each transition, any solution satisfies $(n_i + n_{i+1})/4 \leq |M_i \cup M_{i+1}|$ and hence $\xi(I \cup J) \leq 2 \cdot \mathsf{opt}$. Finally, we obtain the claimed ratio: $\mathsf{apx} \leq 1/2 \cdot \big(2 \cdot \mathsf{opt} + \alpha \cdot \mathsf{opt}\big) = (1 + \alpha/2) \cdot \mathsf{opt}$. □

5

SUMMARY

In this part, we presented the first approximation algorithm for MIM$|_2$ with a tightly analyzed approximation ratio of $1/\sqrt{2\chi}$. It remains open whether any of the problems is APX-hard or if a constant factor approximation for MIM$|_2$ is possible; however, we showed that the latter would imply a constant factor approximation for MIM. We further showed two ways in which MIM and MUM can be approximated by using any algorithm that approximates MIM$|_2$, thereby also presenting the first approximation algorithms for multistage matching problems with an arbitrary number of stages.

As we will see in the next part, the techniques of Algorithm 9 and 13 for MIM can be used as a building block to approximate the newly-introduced set of *preficient* multistage subgraph problems.

Part II

MULTISTAGE SUBGRAPH PROBLEMS

If we want things to stay as they are, things will have to change.
— Giuseppe Tomasi di Lampedusa: *The Leopard* [Tom60]

6 BACKGROUND

This part is based on joint work with Markus Chimani and Tilo Wiedera that has been accepted for publication in the conference proceedings of *XII. Latin-American Algorithms, Graphs and Optimization Symposium* (LAGOS 2023). The NP-hardness proofs in Chapter 8 are not included in the publication.

Subgraph Problems (SPs) are concerned with selecting some feasible set of graph elements (vertices and/or edges) that is optimal w.r.t. some measure. The class of SPs is very rich and includes many traditional problems like Shortest s-t-Path, Minimum s-t-Cut, Maximum Independent Set, Minimum Vertex Cover, Maximum Cut, Maximum Planar Subgraphs, and Steiner Trees.

As problem instances may be subject to change over time, it is often required to solve the same SP multiple times: In a *Multistage Subgraph Problem* (MSP), we ask for an individual, optimal solution per stage, while retaining as much of the previous solution as possible. One example is the problem MIM from Part I, where the underlying SP is the (unweighted) Perfect Matching problem. Depending on the problem, the transition quality may be measured differently. Even if the subgraph problem itself can be solved exactly in polynomial time, this *multistage* variant turns out to be NP-hard in most cases as we have already seen for MIM.

6.1 CONTRIBUTION

In this part, we provide a framework to obtain approximation algorithms for a wide range of multistage subgraph problems, where, following the concept of Part I, we guarantee optimal solutions in each stage. As a key ingredient we define *preficient* (short for "preference efficient") problems (Definition 26); they allow to efficiently compute an optimal solution to an individual stage that prefers some given graph elements. As it turns out, many polynomial-time solvable graph problem are in fact trivially preficient. Our framework algorithm can be applied to any preficient multistage subgraph problem where we measure the transition quality as the number of common graph elements between subsequent stages; it yields an approximation ratio only dependent on the input's intertwinement, see Theorem 27.

A building block of this algorithm, which itself does not depend on the transition quality measure and may therefore be of independent interest, is Theorem 25: Any α-approximation (including $\alpha = 1$) for a t-stage Subgraph

Problem with fixed $t \geq 2$ can be lifted to an approximation for the corresponding unrestricted multistage subgraph problem.

Finally, in Chapter 8, we demonstrate that the class of applicable multistage problems is very rich: It is typically straightforward to construct a preficiency algorithm from classic algorithms. We can thus deduce several new approximation results simply by applying our preficiency framework approximation, without the need of additional deep proofs. As examples, we showcase this for multistage variants of Shortest s-t-Path, Perfect Matching, and Minimum s-t-Cut. Furthermore, several NP-hard (single-stage) problems become polynomial-time solvable on restricted graph classes (e.g., planar, bipartite, etc.); on these, we can apply our framework as well, as we showcase for Maximum Cut, Vertex Cover, and Independent Set.

6.2 FRAMEWORK

All known successful applications of our framework are subgraph problems on graphs. Thus, and for ease of exposition, we will describe our framework solely in this context. However, we will never use any graph-intrinsic properties other than the fact that it is a system of elements. It should be understood that we can in fact replace graphs with any other combinatorial structure (e.g., hypergraphs, matroids, fields, etc.) in all definitions and results, as long as a solution is a subset of elements of said structure.

The following definitions may at first seem overly complicated, but are carefully constructed to be as general as possible, similar to those for general NP-optimization problems, e.g., in [FG06; KV08]. Recall that we use the notation $X(G)$ to refer to the elements of a graph G, i.e., its vertices and edges. An *enriched graph* is a graph with additional information (e.g., weights or labels) at its elements.

Definition 22. *A* Subgraph Problem (SP) *is a combinatorial optimization problem $\mathcal{P} := (\mathbb{G}, f, m, \psi)$, where*

- $\mathbb{G}$ *denotes a class of enriched graphs that is the (in general infinite)* set of possible instances*;*
- *f is a function such that, for an instance $G \in \mathbb{G}$, the set $f(G) \subseteq 2^{X(G)}$ contains the* feasible solutions*; a* feasible solution *$S \in f(G)$ is a subset of $X(G)$;*
- *m is a function such that, for an instance $G \in \mathbb{G}$ and a feasible solution $S \in f(G)$, the* measure *of S is given by $m(G, S)$;*
- *the* goal *ψ is either* min *or* max*.*

Given some instance $G \in \mathbb{G}$, the objective is to find a feasible solution $S \in f(G)$ that is optimal *in the sense that $m(G, S) = \psi\{m(G, S') \mid S' \in f(G)\}$. The* set of optimal solutions *is denoted by $f^*(G)$. An element $x \in X(G)$ is* allowed w.r.t. *$\mathcal{P}$ if it occurs in any optimal solution for the $\mathcal{P}$-instance G, i.e., $x \in \bigcup_{S \in f^*(G)} S =: X_{\mathcal{P}}(G)$.*

As an example, MAXIMUM WEIGHT PERFECT MATCHING is an SP: $\mathbb{G}$ is the class of all edge-weighted graphs, $f(G)$ is the set of all perfect matchings of $G \in \mathbb{G}$, $m(G, S)$ returns the sum of the weights of all edges in S and $\psi = \max$. Similarly, MINIMUM WEIGHT PERFECT MATCHING is an SP with $\psi = \min$, while the plain PERFECT MATCHING problem on unweighted graphs has constant measure m.

Definition 23. *A* Multistage Subgraph Problem (MSP) *is a combinatorial optimization problem $\mathcal{M} = (\mathcal{P}, q)$, where*

- *$\mathcal{P} := (\mathbb{G}, f, m, \psi)$ is a Subgraph Problem;*
- *an instance is a multistage graph $\mathcal{G} = (G_i)_{i \in [\tau]} \in \mathbb{G}^\tau$ for some $\tau > 1$; and*
- *q is a non-negative function such that, given an instance $\mathcal{G} \in \mathbb{G}^\tau$ and subsets $Y_i \subseteq X(G_i)$ and $Y_{i+1} \subseteq X(G_{i+1})$, $q(Y_i, Y_{i+1})$ measures the* transition quality *of these sets for any $i \in [\tau - 1]$.*

Given some instance $\mathcal{G} \in \mathbb{G}^\tau$, let $f^\times(\mathcal{G}) := f^*(G_1) \times \cdots \times f^*(G_\tau)$ denote the set of feasible multistage solutions*, containing τ-tuples of optimal solutions for the individual stages. The objective is to find a feasible multistage solution $\mathcal{S} \in f^\times(\mathcal{G})$ that is maximum w.r.t. q in the sense that $Q(\mathcal{S}) = \max\{Q(\mathcal{S}') \mid \mathcal{S}' \in f^\times(\mathcal{G})\}$ where $Q(\mathcal{S}') := \sum_{i \in [\tau-1]} q(S'_i, S'_{i+1})$ is the* global quality *of $\mathcal{S}$.*

If there is an upper bound t on the number of stages τ, an MSP $\mathcal{M}$ may be denoted by $\mathcal{M}|_t$. MSPs with some fixed function q may be summarily referred to as q-MSPs.

In our definition, we aim at maximizing the transition quality. One could analogously also consider minimizing transition *costs* (see, e.g., [FNS+20; FNR+22; Flu21] or the problem MUM from Part I); however, for clarity of presentation, we opted to solely focus on the former notion. Common choices for transition qualities are the *intersection profit* $q_\cap(S_i, S_{i+1}) := |S_i \cap S_{i+1}|$, e.g., in [FNS+20; BEL+18; BES+21] or measures based on the (symmetric) difference of subsequent stages [BEK21; BET22; KRZ21]. The problem MIM from Part I can be understood as an MSP with the (unweighted) PERFECT MATCHING as the underlying SP and intersection profit as the transition quality.

Considering some SP $\mathcal{P}$ and a multistage graph $\mathcal{G} = (G_i)_{i \in [\tau]}$, we measure the similarity of consecutive stages of $\mathcal{G}$ w.r.t. $\mathcal{P}$ by the *intertwinement* $\chi_\mathcal{P}(\mathcal{G}) := \max_{i \in [\tau-1]} |X_\mathcal{P}(G_i) \cap X_\mathcal{P}(G_{i+1})|$. If the context is clear, we may simply use $\chi := \chi_\mathcal{P}(\mathcal{G})$. Note that, deviating from the intertwinement definition in Part I, this also allows us to count the maximum number of common allowed vertices in two consecutive stages.

We will establish approximation algorithms whose approximation quality decreases monotonously with increasing χ. Consider any MSP $\mathcal{M}$ with polynomial-time solvable SP $\mathcal{P}$ and a 2-stage input graph, where the two stages have nothing in common. Then, optimizing $\mathcal{M}$ is typically a simple matter of solving $\mathcal{P}$ on each stage individually, yielding an exact polynomial-time algorithm. Observe that the intertwinement captures this as $\chi = 0$. Increasing the commonality

between the stages increases both the intertwinement and the multistage problem's complexity, suggesting that intertwinement is a feasible measure. At some tipping point, once stages become very similar, our use of intertwinement loses its expressiveness and other similarity measures should be preferred.

7 ALGORITHMIC TECHNIQUES FOR APPROXIMATION

We first demonstrate how to obtain an approximation algorithm for an MSP with arbitrary many stages, assuming we have an (approximation) algorithm for the MSP with a fixed number of stages. This generalizes Algorithm 13 from Part I in two ways: (i) the generalized method is not restricted to matching problems and (ii) it can make use of an auxiliary algorithm that is capable of handling more than two stages at once. Afterwards, we show that, if the underlying SP has a certain property, we can use an appropriate formulation of Algorithm 9 to get an approximation for the 2-stage case.

7.1 REDUCING THE NUMBER OF STAGES

It is natural to expect that an MSP $\mathcal{M}$ would be easier to solve if the number of stages is bounded by some constant $t \geq 2$. We can show that if we have an α-approximate (or even an exact) algorithm $\mathcal{A}$ for $\mathcal{M}|_t$, one can use it to craft a solution for the unbounded problem that is within factor $\alpha \cdot (t-1)/t$ of the optimum. This result is under the assumption that algorithm $\mathcal{A}$ can also handle smaller instances with $t' \in [t]$ stages; for a single-stage instance it suffices for $\mathcal{A}$ to output any optimal solution. The following Algorithm 24 is a more sophisticated variant of Algorithm 13, where we use an approximation for 2-stage instances to obtain a solution for instances with an arbitrary number of stages.

Informally, $\mathcal{G}|_i^r$ denotes the subinstance of $\mathcal{G}$ with r stages, starting at the ith stage. Formally, for $i \in [\tau]$, let $\mathcal{G}|_i^r$ consist exactly of the stages with index in the range $[i, \min\{i+r-1, \tau\}]$. Let $\mathcal{M}$ be an MSP, $t \geq 2$, and assume we have an α-approximation algorithm $\mathcal{A}$ for $\mathcal{M}|_t$. Algorithm 24 constructs a set $\{\mathcal{S}_k \mid k \in [t]\}$ of candidate multistage solutions and returns one with maximum global quality. Each candidate solution $\mathcal{S}_k$ is built as follows: Start with a partial solution obtained from calling $\mathcal{A}$ on the first k stages of $\mathcal{G}$; then iteratively consider the subsequent t stages as a subinstance, compute a partial solution using $\mathcal{A}$ and append it to the existing partial solution (denoted by operator $\circ$); repeat this step until eventually stage τ has been considered (in general, the final subinstance containing stage τ may again consist of less than t stages). Figure 7.1 shows an example of the algorithm's subinstance calls.

Theorem 25. *Let $\mathcal{M} = (\mathcal{P}, q)$ be an MSP and $\mathcal{A}$ an α-approximation for $\mathcal{M}|_t$ for some fixed $t \geq 2$ (possibly with $\alpha = 1$). Then Algorithm 24 yields a β-approximation for $\mathcal{M}$, where $\beta := \alpha(t-1)/t$.*

Proof. The algorithm's output $\mathcal{S}$ is the candidate with optimal profit and thus has at least average profit over all t candidate solutions: $Q(\mathcal{S}) \geq 1/t \cdot \sum_{k \in [t]} Q(\mathcal{S}_k)$.

Algorithm 24: Approximation of an MSP $\mathcal{M} = (\mathcal{P}, q)$, given an α-approximation $\mathcal{A}$ for $\mathcal{M}|_t$

Input: Enriched multistage graph $\mathcal{G} = (G_1, \dots, G_\tau)$, α-approximation $\mathcal{A}$ for $\mathcal{M}|_t$
Output: Multistage solution $\mathcal{S}$

```
1 S = (∅, ..., ∅)
2 foreach k ∈ [t] do
3 |   S_k ← A(G|_1^k)
4 |   i ← k + 1
5 |   while i ≤ τ do
6 |   |   S_k ← S_k ∘ A(G|_i^t)
7 |   |   i ← i + t
8 |   if Q(S_k) ≥ Q(S) then S ← S_k
9 return S
```

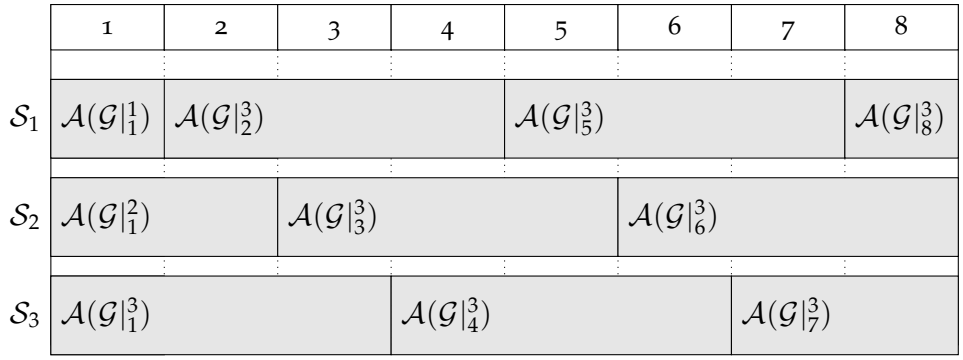


Figure 7.1: Example of the subinstances that Algorithm 24 computes to obtain candidate solutions $\mathcal{S}_1, \mathcal{S}_2, \mathcal{S}_3$ for an MSP with overall $\tau = 8$ stages. Here, the auxiliary algorithm $\mathcal{A}$ provides (approximate) solutions for up to $t = 3$ stages.

Let $I_k := \{b \leq \tau \mid j \in \mathbb{N} \colon b = (k+1) + j \cdot t\}$ denote the set of values that i takes in the kth iteration of the foreach loop. For $r \in \mathbb{N}_{>0}$, let $Q(\mathcal{S}_k|_i^r) := \sum_{j \in [i, i+r-2]} q(S_j, S_{j+1})$ denote the quality of $\mathcal{S}_k := (S_j)_{j \in [\tau]}$ restricted to $\mathcal{G}|_i^r$. As transition qualities are non-negative, we have $Q(\mathcal{S}_k) \geq Q(\mathcal{S}_k|_1^k) + \sum_{i \in I_k} Q(\mathcal{S}_k|_i^t)$ and thus $Q(\mathcal{S}) \geq 1/t \cdot \sum_{k \in [t]} \left(Q(\mathcal{S}_k|_1^k) + \sum_{i \in I_k} Q(\mathcal{S}_k|_i^t)\right)$.

Let $Q^*|_r^i$ be the optimal quality achievable for $\mathcal{G}|_i^r$. Since $\mathcal{A}$ is an α-approximation for $\mathcal{M}|_t$, we have for all $k \in [t]$ that $Q(\mathcal{S}_k|_1^k) \geq \alpha Q^*|_k^1$ and also $Q(\mathcal{S}_k|_i^t) \geq \alpha Q^*|_t^i$ for all $i \in I_k$. By construction, $[2, \tau]$ is the disjoint union of all I_k and thus

$$Q(\mathcal{S}) \geq 1/t \cdot \sum_{k \in [t]} \left(\alpha Q^*|_k^1 + \sum_{i \in I_k} \alpha Q^*|_t^i\right) = \alpha/t \cdot \Big(\sum_{k \in [t]} Q^*|_k^1 + \sum_{i \in [2,\tau]} Q^*|_t^i \Big).$$

Let $\mathcal{S}^* := (S_j^*)_{j\in[\tau]}$ be an optimal multistage solution. As by definition $Q^*|_r^i \geq Q(\mathcal{S}_k^*|_i^r)$, we have

$$Q(\mathcal{S}) \geq \alpha/t \cdot \big(\underbrace{\sum_{k\in[t]} Q(\mathcal{S}_k^*|_1^k)}_{(a)} + \underbrace{\sum_{i\in[2,\tau]} Q(\mathcal{S}_k^*|_i^t)}_{(b)} \big).$$

For each fixed $j \in [t-1]$, the term $q(S_j^*, S_{j+1}^*)$ appears exactly $t-j$ times in (a) (namely once for each $k \in [j+1, t]$) and exactly $j-1$ times in (b) (namely once for each $i \in [2, j]$). Considering any larger $j \in [t, \tau-1]$, the term $q(S_j^*, S_{j+1}^*)$ does not appear in (a) and exactly $t-1$ times in (b) (namely once for each $i \in [j-t+2, j]$). We thus have

$$\sum_{k\in[t]} Q(\mathcal{S}_k^*|_1^k) + \sum_{i\in[2,\tau]} Q(\mathcal{S}_k^*|_i^t) = (t-1) \cdot \sum_{j\in[\tau-1]} q(S_j^*, S_{j+1}^*) = (t-1)Q(\mathcal{S}^*)$$

and can conclude $Q(\mathcal{S}) \geq \alpha(t-1)/t \cdot Q(\mathcal{S}^*)$. □

7.2 APPROXIMATING TWO STAGES

Building on Algorithm 9 from Part I, we present an algorithm that computes an approximate solution for the 2-stage restriction of any $q_\cap$-MSP where the underlying SP has a certain property.

Definition 26. *An SP $\mathcal{P} := (\mathbb{G}, f, m, \psi)$ is called* preficient *(short for* preference efficient*) if there is an algorithm $\mathcal{B}(G, Z)$ that solves the following problem in polynomial time: Given a graph $G \in \mathbb{G}$ and subset $Z \subseteq X(G)$, compute an optimal solution $S = \operatorname{argmax}_{S'\in f^*(G)} |S' \cap Z|$. Such an algorithm $\mathcal{B}$ is called a* preficiency algorithm *for $\mathcal{P}$. An MSP is called preficient if its underlying SP is preficient.*

Note that for a preficient SP $\mathcal{P}$, the set $X_\mathcal{P}(G)$ of allowed elements is trivially computable in polynomial time: a graph element $x \in X(G)$ is allowed w.r.t. $\mathcal{P}$ if and only if it is in a solution computed by $\mathcal{B}(G, \{x\})$.

Provided we have a preficiency algorithm $\mathcal{B}$ for an SP $\mathcal{P}$, the following algorithm is an approximation for $\mathcal{M}|_2$ where $\mathcal{M} = (\mathcal{P}, q_\cap)$ (see Algorithm 28 for pseudocode):

Given a 2-stage graph $\mathcal{G} = (G_1, G_2)$, we generate candidate 2-stage solutions in a loop while storing the currently best overall solution throughout. In the loop, with iterations indexed by $i = 1, 2, \ldots$, we consider a set Y that keeps track of *intersection elements* which have not been in a solution for G_1 in any previous iteration; we initialize Y with $X_\cap := X_\mathcal{P}(G_1) \cap X_\mathcal{P}(G_2)$ and update Y at the beginning of each iteration. In iteration i, we use $\mathcal{B}$ to find some solution $S_1^{(i)} \in f^*(G_1)$ that optimizes $q_\cap(S_1^{(i)}, Y)$; we then use $\mathcal{B}$ again to find a second solution $S_2^{(i)} \in f^*(G_2)$ that optimizes $q_\cap(S_1^{(i)}, S_2^{(i)})$. The loop stops as soon as Y is empty; the output is the 2-stage solution (S_1, S_2) with maximum quality over all candidate solutions.

The approximation ratio depends on the input's intertwinement $\chi = |X_\cap|$.

Algorithm 28: Approximation of $\mathcal{M}|_2$ for an MSP $\mathcal{M} = (\mathcal{P}, q_\cap)$ with preficient $\mathcal{P}$

Input: Enriched 2-stage graph $\mathcal{G} = (G_1, G_2)$ with non-empty intersection, preficiency algorithm $\mathcal{B}$ for $\mathcal{P}$
Output: 2-stage solution $\mathcal{S} = (S_1, S_2)$

$(S_1, S_2) \leftarrow (\varnothing, \varnothing)$
for $i = 1, 2, \dots$ **do**
  $Y \leftarrow X_\cap \setminus \bigcup_{j \in [i-1]} S_1^{(j)}$
  if $Y = \varnothing$ **then return** (S_1, S_2)
  $S_1^{(i)} \leftarrow \mathcal{B}(G_1, Y)$
  $S_2^{(i)} \leftarrow \mathcal{B}(G_2, S_1^{(i)})$
  if $q_\cap(S_1^{(i)}, S_2^{(i)}) \geq q_\cap(S_1, S_2)$ **then** $(S_1, S_2) \leftarrow (S_1^{(i)}, S_2^{(i)})$

Theorem 27. *Consider an MSP $\mathcal{M} = (\mathcal{P}, q_\cap)$ with preficient $\mathcal{P}$. Then, Algorithm 28 is a polynomial-time $1/\sqrt{2\chi}$-approximation algorithm for $\mathcal{M}|_2$.*

The following proof is analogous to the proof of Lemma 11, differing only in the use of more generalized terms ("perfect matching" becomes "solution", "edges" becomes "elements", etc.) and notation that relates to the pseudocode presented here. However, to enable a coherent understanding of Algorithm 28, we will fully restate the proof here completely.

Proof. Let $\mathcal{B}$ be a preficiency algorithm for $\mathcal{P}$ and $\mathcal{G}$ a 2-stage graph. We can assume w.l.o.g. that $X_\cap$ is non-empty and thus $\mathsf{opt} \geq 1$. Clearly, the first iteration establishes $\mathsf{apx} \geq 1$.

In each iteration i of the loop, at least one element of $X_\cap$ that has not been in any previous first stage solution is contained in a solution for G_1 (otherwise the loop terminates) and hence the loop terminates in polynomial time. Let k denote the number of iterations. For any $i \in [k]$, let $(S_1^{(i)}, S_2^{(i)})$ denote the 2-stage solution computed in the ith iteration. Let (S_1^*, S_2^*) denote an optimal 2-stage solution and $S_\cap^* := S_1^* \cap S_2^* \subseteq X_\cap$ its intersection (note that $S_1^* \cap X_\cap \setminus S_\cap^*$ may be non-empty). Let $R_i := (S_1^{(i)} \cap X_\cap) \setminus \bigcup_{j \in [i-1]} R_j$ denote the set of intersection elements that are in $S_1^{(i)}$ but not in $S_1^{(j)}$ for any previous iteration $j < i$; let $r_i := |R_i|$. Note that in iteration i, the algorithm first searches for a solution $S_1^{(i)} \in f^*(G_1)$ that maximizes

$$q_\cap(S_1^{(i)}, X_\cap \setminus \bigcup_{j \in [i-1]} S_1^{(j)}) = |S_1^{(i)} \cap (X_\cap \setminus \bigcup_{j \in [i-1]} R_j)| = r_i.$$

We define $R_i^* := (S_1^{(i)} \cap S_\cap^*) \setminus \bigcup_{j \in [i-1]} R_j^*$ and $r_i^* := |R_i^*|$ equivalently to R_i, but w.r.t. $S_\cap^*$ instead of $X_\cap$ (cf. Figure 4.2 for a visualization of similar set inclusions). Thus, R_i^* contains those elements of $S_\cap^*$ that are selected (into $S_1^{(i)}$) for the first time over all iterations. Observe that $R_i \cap S_\cap^* = R_i^*$.

Let $x := \sqrt{2\chi}$. For every $i \in [k]$, the algorithm chooses $S_2^{(i)}$ that maximizes $q_\cap(S_1^{(i)}, S_2^{(i)})$. Since we may choose $S_2^{(i)} = S_2^*$, it follows that $\mathsf{apx} \geq \max_{i\in[k]} r_i^*$. Thus, if $\max_{i\in[k]} r_i^* \geq \mathsf{opt}/x$, we have a $1/x$-approximation. In case $\mathsf{opt} \leq x$, any solution with profit at least 1 yields a $1/x$-approximation (which we trivially achieve as discussed above). We show that we are always in one of the two cases.

Assume that $\mathsf{opt} > x$ and simultaneously $r_i^* < \mathsf{opt}/x$ for all $i \in [k]$. Since we distribute $S_\cap^*$ over the disjoint sets $\{R_i^* \mid i \in [k]\}$, each containing less than opt/x elements, we know that $k > x$ (thus $k \geq \lceil x \rceil =: \bar{x}$). Recall that in iteration i, $Y = X_\cap \setminus \bigcup_{j\in[i-1]} S_1^{(j)}$. Thus, we have that $Y \cap \bigcup_{j\in[i-1]} R_j$ is empty and the number of elements of S_1^* that are counted towards $q(S_1^{(i)}, Y) = r_i$ is

$$\big|(S_1^* \cap X_\cap) \setminus \bigcup_{j\in[i-1]} R_j\big| \geq \big|S_\cap^* \setminus \bigcup_{j\in[i-1]} R_j\big| = \big|S_\cap^* \setminus \bigcup_{j\in[i-1]} R_j^*\big|.$$

Hence, the latter term is a lower bound on r_i and we deduce:

$$\begin{aligned} r_i &\geq \big|S_\cap^* \setminus \bigcup_{j\in[i-1]} R_j^*\big| = \mathsf{opt} - \sum_{j\in[i-1]} r_j^* \\ &\overset{(\star)}{\geq} \mathsf{opt} - \sum_{j\in[i-1]} \frac{\mathsf{opt}}{x} = \mathsf{opt} \cdot \big(1 - \frac{i-1}{x}\big) \geq \bar{x}\big(1 - \frac{i-1}{x}\big). \end{aligned}$$

Thereby, strict inequality holds at $(\star)$ for $i \geq 2$. This raises a contradiction:

$$\begin{aligned} \chi = |X_\cap| &= \big|\bigcup_{i\in[k]} R_i\big| \geq \sum_{i\in[\bar{x}]} r_i \gneqq \sum_{i\in[\bar{x}]} \bar{x}\big(1 - \frac{i-1}{x}\big) \\ &= \bar{x} \cdot \big(\sum_{i\in[\bar{x}]} 1 - \sum_{i\in[\bar{x}-1]} \frac{i}{x}\big) = \bar{x}\big(\bar{x} - \frac{(\bar{x}-1)\bar{x}}{2x}\big) \\ &= \bar{x}^2\big(1 - \frac{\bar{x}-1}{2x}\big) \geq \bar{x}^2\big(1 - \frac{x}{2x}\big) = \frac{\bar{x}^2}{2} \geq \frac{x^2}{2} = \chi. \end{aligned}$$

□

As we know from Theorem 7, there is (weak) evidence that a χ-dependent ratio may be unavoidable for the Multistage Perfect Matching problem. This evidence carries over to the generalized algorithm, as the generalized problem cannot be treated better than any of its special cases:

Remark 29. *The analysis in Theorem 27 is* tight *in the sense that Algorithm 28 cannot guarantee a better approximation ratio for arbitrary preficient SPs. This is due to the fact that we know from Lemma 10 that there is an instance family for the Multistage Perfect Matching Problem for which the stripped down version of Algorithm 28 yields precisely this ratio. This does not rule out that for some "simpler" SP, our algorithm may achieve a better approximation ratio.*

8

APPLICATIONS

As examples of the utility of our framework, we show preficiency for a variety of SPs and NP-hardness for the corresponding $q_\cap$-MSPs. Proving preficiency typically always follows the same pattern for these problems: we modify (or assign) weights for those graph elements that are to be preferred in a way that does not interfere with the feasibility of a solution; we then apply a polynomial algorithm (as a black box) to solve the problem w.r.t. the modified weights. The above Theorems 25 and 27 then allow us to deduce approximation algorithms for the corresponding $q_\cap$-MSP.

In general, instead of manipulating the weights, one could (try to) carefully manipulate the arithmetic computations in the black box algorithm. This then would typically also allow to consider non-negative weights (instead of strictly positive ones). However, we refrain from doing so herein for clarity of exposition and general applicability to any black box algorithm where such arithmetic modifications may not be straightforward.

Under the new framework, the following preficiency proofs are consistently remarkably short. In fact, we are able to use nearly identical proofs for a range of $q_\cap$-MSP formulations of classical combinatorial problems. To justify the need for approximation algorithms, we also present or refer to existing proofs that the presented multistage problems are NP-hard. More precisely, all of them are even NP-hard when restricted to two stages, using very similar proof ideas. We stress that there are no known constant-ratio approximations for any of the MSPs.

WELL-BEHAVED WEIGHT MODIFICATIONS. Let $G = (V, E)$ be a graph with weights $w\colon X \to \mathbb{N}_{>0}$ on its elements. Let $w(Z) := \sum_{e\in Z} w(e)$ denote the weight of a subset $Z \subseteq X$. Given an element subset $Y \subseteq X$ and some $\varepsilon \in \mathbb{Q}$, we define the *modified weight function* $w_{\varepsilon,Y}\colon X \to \mathbb{Q}$ that is identical to w on $X \setminus Y$ and $w_{\varepsilon,Y}(e) := w(e) - \varepsilon$ for $e \in Y$.

Consider some SP P. The modified weight function $w_{\varepsilon,Y}$ is *well-behaved w.r.t. P* if the following properties hold for any two edge sets $Z, Z' \subseteq X$:

Positivity: $w_{\varepsilon,Y}(e) > 0$ for all $e \in X$.

Monotonicity: If $w(Z') < w(Z)$, then $w_{\varepsilon,Y}(Z') < w_{\varepsilon,Y}(Z)$.

Preference: If $w(Z') = w(Z)$ and $|Z' \cap Y| > |Z \cap Y|$, then: if P is a minimization problem then $w_{\varepsilon,Y}(Z') < w_{\varepsilon,Y}(Z)$; otherwise ($P$ is a maximization problem) $w_{\varepsilon,Y}(Z') > w_{\varepsilon,Y}(Z)$.

Naturally, we will choose $\varepsilon > 0$ for minimization problems and $\varepsilon < 0$ for maximization problems.

8.1 MULTISTAGE MINIMUM WEIGHT PERFECT MATCHING

As a first example we consider a multistage version of MINIMUM WEIGHT PERFECT MATCHING. In a graph $G = (V, E)$ with positive edge weights $w\colon E \to \mathbb{N}_{>0}$, an edge set $F \subseteq E$ is a *perfect matching* if each $v \in V$ is incident with exactly one edge in F. A perfect matching F has *minimum weight* if for each perfect matching F' we have $w(F') \geq w(F)$.

Definition 30 (MMINPM). *Given a multistage graph $(G_i)_{i\in[\tau]}$ with positive edge weights $w_i\colon E_i \to \mathbb{N}_{>0}$ for each $i \in [\tau]$, find a $q_\cap$-optimal multistage solution $(F_i)_{i\in[\tau]}$ such that for each $i \in [\tau]$, $F_i \subseteq E_i$ is a minimum weight perfect matching w.r.t. w_i in G_i.*

MMINPM$|_2$ is NP-hard as the special case of uniform edge weights is equivalent to MIM (Theorem 6).

Theorem 31. *There is a $1/\sqrt{8\chi}$-approximation algorithm for* MMINPM *and a $1/\sqrt{2\chi}$-approximation for* MMINPM$|_2$.

Proof. We only need to show preficiency for MMINPM and apply Theorems 25 and 27. Let $G = (V, E)$ be a graph with edge weights $w\colon E \to \mathbb{N}_{>0}$ and $Y \subseteq E$ the set of edges to be preferred. Set $\varepsilon := 1/(|E| + 1)$. We use an arbitrary polynomial-time algorithm for computing a minimum weight perfect matching in G with the modified weight function $w_{\varepsilon,Y}$ (e.g., Edmond's blossom algorithm [Edm65b; LP86]) and denote its output by F. Since $w_{\varepsilon,Y}$ is well-behaved (which can be checked by straightforward computation), F is a minimum weight perfect matching in G such that $|F \cap Y|$ is maximum. □

This result does not contradict the linear lower bound on the approximation ratio discussed in [BEL+18], since they (i) minimize an objective function combining transition costs and per-stage solution quality and (ii) do not consider intertwinement dependency. Note that there are graphs with linear intertwinement $\chi \in \Theta(|E|)$.

8.2 MULTISTAGE SHORTEST s-t-PATH

The classic problem of finding a shortest s-t-path is easily formulated as an SP when we view a path as a set of edges. The corresponding MSP is introduced and shown to be NP-hard already for 2-stage directed acyclic graphs in [FNS+20] (although they consider a slightly different definition, the NP-hardness proof directly translates to our formulation).

In a graph $G = (V, E)$ with edge weights $w\colon E \to \mathbb{N}_{>0}$ and two terminal vertices $s, t \in V$, an edge set $F \subseteq E$ is an *s-t-path* in G if it is of the form $\{v_1v_2, v_2v_3, \ldots, v_{k-1}v_k\}$ where $k \geq 2$, $v_1 = s$ and $v_k = t$. An s-t-path is a *shortest s-t-path* if for each s-t-path F' we have $w(F) \leq w(F')$.

Definition 32 (MSPATH). *Given a multistage graph $(G_i)_{i\in[\tau]}$ with positive edge weights $w_i\colon E_i \to \mathbb{N}_{>0}$ for each $i \in [\tau]$ and terminal vertices $s, t \in V_i$, find*

a $q_\cap$-optimal multistage solution $(F_i)_{i\in[\tau]}$ such that for each $i \in [\tau]$, $F_i \subseteq E_i$ is a shortest s-t-path in G_i.

From a practical viewpoint, one might consider a fixed query to be a restriction. However, allowing a different query (s_i, t_i) at each stage would not change the problem, as we can easily obtain an equivalent instance where the terminals are chosen consistently: Add two new vertices (s, t) to all stages, together with paths of length 2 (via new vertices) from s to s_i as well as from t_i to t in the corresponding stage. This reduction does not affect approximation ratios, since the new edges are only ever present in a single stage.

Theorem 33. *There is a $1/\sqrt{8\chi}$-approximation algorithm for* MSPATH *and a $1/\sqrt{2\chi}$-approximation for* MSPATH$|_2$.

Proof. Follow the proof of Theorem 31 using, e.g., Dijkstra's algorithm [Dij59] to compute a shortest s-t-path in polynomial time. □

8.3 MULTISTAGE MINIMUM s-t-CUT

Another classical SP is MINIMUM s-t-CUT. While it is usually defined as a selection of vertices, we will first discuss the notion where we want to select a set of edges that separates the target vertices. Then, we investigate two multistage variants that select vertices.

In a graph $G = (V, E)$ with edge weights $w\colon E \to \mathbb{N}_{>0}$, two vertices $s, t \in V$, an edge set $F \subseteq E$ is an *s-t-cut* if there is no s-t-path in $(V, E \setminus F)$. An s-t-cut F is *minimum* if for each s-t-cut F' we have $w(F') \geq w(F)$.

Definition 34 (MMINCUT)**.** *Given a multistage graph $(G_i)_{i\in[\tau]}$ with positive edge weights $w_i\colon E_i \to \mathbb{N}_{>0}$ and terminal vertices $s_i, t_i \in V_i$ for each $i \in [\tau]$, find a $q_\cap$-optimal multistage solution $(F_i)_{i\in[\tau]}$ such that for each $i \in [\tau]$, $F_i \subseteq E_i$ is a minimum s_i-t_i-cut for s_i, t_i in G_i.*

Theorem 35. MMINCUT$|_2$ *is NP-hard, even if $s_1 = s_2$, $t_1 = t_2$, and the edges have uniform weights.*

Proof. We will perform a reduction from the NP-hard MAXIMUM CUT [GJ79] to the decision variant of MMINCUT: Given G, $(s_i, t_i)_{i\in[\tau]}$ and a number $\kappa \in \mathbb{N}$, is there a $q_\cap$-optimal multistage solution for MMINCUT with profit at least κ? In MAXIMUM CUT, one is given an undirected graph $G = (V, E)$ and a number $k \in \mathbb{N}$; the question is whether there is a vertex set $U \subseteq V$, such that $|\delta(U)| \geq k$. In the first stage, we will construct a bundle of s-t-paths for each vertex of the original graph and in the second stage we will create two s-t-paths for each edge (cf. Figure 8.1). A minimum s-t-cut in the first stage will correspond to a vertex selection and a minimum s-t-cut in the second stage will allow us to count the number of edges that are incident to exactly one selected vertex.

Given an instance $\mathcal{I} := \big(G = (V, E), k\big)$ of MAXIMUM CUT, we will construct an equivalent instance $\mathcal{J} := \big(\mathcal{G}, (s_i, t_i)_{i\in[\tau]}, \kappa\big)$ for MMINCUT. Set $\kappa := |E| + k$.

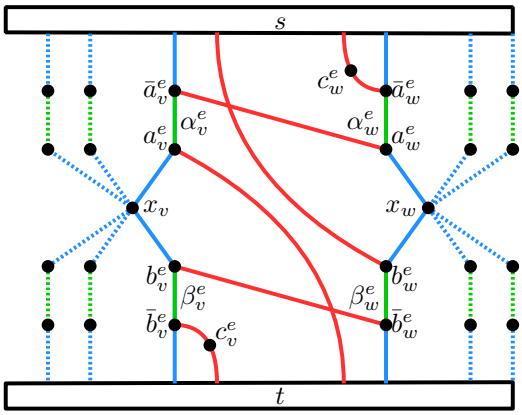


Figure 8.1: Two vertex gadgets and one edge gadget, as used in the proof of Theorem 35. Vertices s and t are enlarged. Edges in $E_\cap$ are bold and green, edges in $E_1 \setminus E_\cap$ are blue, edges in $E_2 \setminus E_\cap$ are red.

Start with a 2-stage graph $\mathcal{G} := (G_1, G_2)$ and vertices $s, t \in V_1 \cap V_2$ that are used as terminals s_i and t_i in both stages.

In the first stage, create a vertex x_v for each $v \in V$. For each $v \in V$ and for each edge $e \in \delta(v) := \{e \in E \mid v \in e\}$, create a path of length 3 from s to x_v whose middle edge is labeled α_v^e and a path of length 3 from x_v to t whose middle edge is labeled β_v^e. Let $A_v := \{\alpha_v^e \mid e \in \delta(v)\}$ and $B_v := \{\beta_v^e \mid e \in \delta(v)\}$. Let a_v^e (b_v^e) denote the endpoint of α_v^e (β_v^e) closer to x_v and $\bar{a}_v^e$ ($\bar{b}_v^e$) the other one.

The second stage reuses all a-, $\bar{a}$-, b- and $\bar{b}$-vertices and all α- and β-edges. For each edge $e = vw \in E$, add two vertices c_v^e, c_w^e. By adding the edge sets $\{sc_w^e, c_w^e\bar{a}_w^e, a_w^e\bar{a}_v^e, a_v^e t\}$ and $\{sb_w^e, \bar{b}_w^e b_v^e, \bar{b}_v^e c_v^e, c_v^e t\}$, we construct two paths, A^e and B^e, of length 6 from s to t: A^e via the α-edges and B^e via the β-edges. The c-vertices are there to avoid unwanted edges in $E_\cap$.

Since for each $v \in V$, a minimum cut C_1 in G_1 needs to cut all s-t-paths through x_v, C_1 contains exactly $|\delta(v)|$ edges from these paths; these are either all part of the 3-paths from s to x_v or all part of the 3-paths from x_v to t.

Claim. *$\mathcal{J}$ is a yes-instance if and only if $\mathcal{I}$ is a yes-instance.*

Proof of Claim. We prove the two implications separately:

"$\Leftarrow$" Suppose there is some $S \subseteq V$, such that $|\delta(S)| \geq k$. For each $v \in S$, add A_v to C_1 and for each $v \in V \setminus S$, add B_v to C_1. Thus, C_1 is a minimum s-t-cut in G_1. For $e = vw \in \delta(S)$ assume w.l.o.g. $v \in S$ and add α_v^e and β_w^e to C_2. For $e = vw \in E \setminus \delta(S)$, add arbitrarily either $\{\alpha_v^e, \beta_w^e\}$ or $\{\alpha_w^e, \beta_v^e\}$ to C_2. Thus, C_2 is a minimum s-t-cut in G_2.

Consider some edge $e = vw \in E$. If $e \in \delta(S)$, assume w.l.o.g. $v \in S$ and $w \in V \setminus S$. Then, $C_1 \cap C_2$ contains two edges of $A^e \cup B^e$, namely α_v^e and β_w^e. If $e \notin \delta(S)$, $C_1 \cap C_2$ contains exactly one edge of $A^e \cup B^e$. Thus, $|C_1 \cap C_2| = |\biguplus_{e \in E} C_1 \cap C_2 \cap (A^e \cup B^e)| = |E| + |\delta(S)|$.

"⇒" Let $\mathcal{C} := (C_1, C_2)$ be a multistage minimum s-t-cut with large intersection $|C_1 \cap C_2| \geq |E| + k$. For $e \in E$, let $m_e := |C_1 \cap C_2 \cap (A^e \cup B^e)| \leq 2$. Observe that, by construction of G_2, $|C_1 \cap C_2| = \sum_{e \in E} m_e$ Thus, by pigeonhole principle, there are at least k edges with $m_e = 2$.

Recall that w.l.o.g. and by optimality of the individual stages, we can assume for each $v \in V$ that C_1 contains either all of A_v or all of B_v but no elements of the other. This yields a selection $U \subseteq V$: Select $v \in V$ into U if and only if $A_v \subseteq C_1$. We observe that $m_e = 2$ then induces $e \in \delta(U)$ and we obtain $|\delta(S)| \geq k$. ◁

This concludes the proof. □

Theorem 36. *There is a $1/\sqrt{8\chi}$-approximation algorithm for* MMinCut *and a $1/\sqrt{2\chi}$-approximation for* MMinCut$|_2$.

Proof. Follow the proof of Theorem 31 using, e.g., the algorithm by Ford and Fulkerson [FF56] to compute a minimum s-t-cut in polynomial time. □

VERTEX VARIANTS. The problem of finding a minimum s-t-cut for each stage can also be optimized to maintain the same set of vertices that are connected to s. For a concise problem definition, we need new terminology: in a graph $G = (V, E)$ with edge weights $w\colon E \to \mathbb{N}_{>0}$ and vertices $s, t \in V$, a vertex set $S \subseteq V$ with $s \in S$, $t \notin S$ is an *s-t-separating partition*; S is *optimal*, if the induced s-t-cut $\delta(S) = \{e \in E \mid |e \cap S| = 1\}$ has minimum weight.

Given a multistage graph $(G_i)_{i \in [\tau]}$ with edge weights $w_i\colon E_i \to \mathbb{N}_{>0}$ and terminal vertices $s_i, t_i \in V_i$ for each $i \in [\tau]$, we may ask for a $q_\cap$-optimal multistage solution $(S_i)_{i \in [\tau]}$ such that for each $i \in [\tau]$, $S_i \subseteq V_i$ is an optimal s_i-t_i-separating partition in G_i. A natural variation is to consider the quality function $q' := |S_i \cap S_{i+1}| + |(V_i \setminus S_i) \cap (V_{i+1} \setminus S_{i+1})|$ instead of $q_\cap$.

Theorem 37. *Both vertex variants of* MMinCut *(using $q_\cap$ or q', resp.) are polynomial-time solvable.*

Proof. Consider $q_\cap$. For each stage i, choose the cardinality-wise largest optimal s_i-t_i-separating partition S_i. Observe that S_i with $s_i \in S_i$ is unique, since every other optimal s_i-t_i-separating partition S_i' is a strict subset of S_i. Clearly, S_i can be found in polynomial time. Since we optimize the intersection of the stage-wise partitions, we obtain a global optimum by having each S_i maximal.

Consider q'. The problem can be easily reduced to a single-stage minimum s-t-cut problem as shown in [BEK21]: Add disjoint copies of each stage to an empty graph, and two new vertices s^* and t^*. Add an edge with infinite weight from s^* to each s_i and one from each t_i to t^*. For each occurrence of a vertex in two adjacent stages, add an edge with small positive weight ε between the two vertex copies. A minimum s^*-t^*-cut in this graph directly induces a minimum s_i-t_i-cut in each stage such that the number of vertices that are in $(S_i \cup S_{i+1}) \setminus (S_i \cap S_{i+1})$ is minimized and thus q' is maximized. □

8.4 MULTISTAGE WEAKLY BIPARTITE MAXIMUM CUT

In contrast to the SPs considered before, the MAXIMUM CUT problem has a maximization goal. We will see that this requires a different technique for the NP-hardness result, but proving preficiency is essentially the same as before.

In a graph $G = (V, E)$ with edge weights $w\colon E \to \mathbb{N}_{>0}$, a vertex set $U \subseteq V$ induces a *maximum cut* $\delta(U) = \{e \in E \mid |e \cap U| = 1\}$ if for each vertex set $U' \subseteq V$ we have $w(\delta(U')) \leq w(\delta(U))$. The (unfortunately named) class of *weakly bipartite graphs* is defined in [GP81] and contains in particular also planar and bipartite graphs. While the precise definition is not particularly interesting to us here, we make use of the fact that, although MAXIMUM CUT is NP-hard in general [Yan78], a maximum cut can be computed in polynomial time on weakly bipartite graphs [GP81].

Definition 38 (MWBMAXCUT)**.** *Given a multistage graph $(G_i)_{i\in[\tau]}$ with edge weights $w_i\colon E_i \to \mathbb{N}_{>0}$ for each $i \in [\tau]$ where each stage is weakly bipartite, find a $q_\cap$-optimal multistage solution $(F_i)_{i\in[\tau]}$ such that for each $i \in [\tau]$, $F_i \subseteq E_i$ is a maximum cut in G_i.*

The NP-hardness proof for MWBMAXCUT uses a similar technique as before.

Theorem 39. MWBMAXCUT$|_2$ *is NP-hard already on* 2*-stage graphs where both stages are weakly bipartite.*

Proof. We will perform a reduction from the NP-hard unweighted MAXIMUM CUT problem on graphs with maximum degree 3 [Yan78] to the decision variant of MWBMAXCUT: "Given a two-stage graph $\mathcal{G}$ with edge weights $w_i\colon E \to \mathbb{N}_{>0}$ for $i \in [2]$ where each stage is weakly bipartite and a number $\kappa \in \mathbb{N}$, is there a $q_\cap$-optimal multistage solution (F_1, F_2) for MWBMAXCUT with intersection profit $q_\cap(F_1, F_2) \geq \kappa$?" In MAXIMUM CUT, one is given an undirected graph $G = (V, E)$ and a natural number k; the question is to decide whether there is an $S \subseteq V$ such that $|\delta(S)| \geq k$.

Given an instance $\mathcal{I} := (G = (V, E), k)$ of MAXIMUM CUT, we construct an equivalent instance $\mathcal{J} := (\mathcal{G}, \kappa)$ of MWBMAXCUT. Set $\kappa := |E| + k$. We start with an empty 2-stage graph $\mathcal{G} := (G_1, G_2)$. In G_1, we will have disjoint planar gadgets for each vertex, allowing three incident edges each. In G_2, we will have disjoint planar gadgets for each edge, intersecting with the two corresponding vertex gadgets in G_1 (cf. Figure 8.2). A maximum cut in G_1 will correspond to a vertex selection in G and a maximum cut in G_2 will allow us to count the edges of G that are incident to exactly one selected vertex.

In G_1, we create a gadget for each vertex $v \in V$ as follows: Add a 9-cycle and label its vertices counter-clockwise with $a^e_v, b^e_v, c^e_v, a^{e'}_v, b^{e'}_v, c^{e'}_v, a^{e''}_v, b^{e''}_v, c^{e''}_v$, where $\{e, e', e''\} = \delta(v)$ are the edges incident with v (if $|\delta(v)| < 3$, the vertices are labeled with fictional indices). Add a vertex x_v and edges $\{x_v b^e_v \mid e \in \delta(v)\}$, as well as edges $\{a^e_v c^e_v \mid e \in \delta(v)\}$.

In G_2, we start with the same vertex set except for the x-vertices and create a gadget for each $e = vw \in E$ as follows: Add edges $a^e_v c^e_w, b^e_v b^e_w, c^e_v a^e_w$ and for each $u \in \{v, w\}$, add a path of length 3 between $a_{u,e}$ and $c_{u,e}$ using new inner

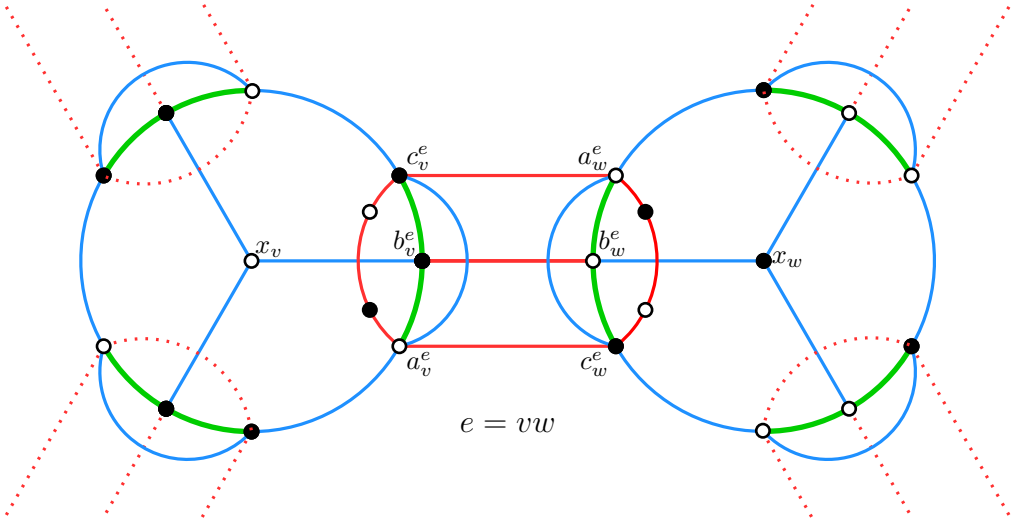


Figure 8.2: Two vertex gadgets and one edge gadget as used in the proof of Theorem 39: Edges in $E_\cap$ are bold and green, edges in $E_1 \setminus E_\cap$ are blue, and edges in $E_2 \setminus E_\cap$ are red. The white vertices display an optimal vertex selection $S_1 \cup S_2$.

vertices. Also, the edges $a_u^e b_u^e$ and $b_u^e c_u^e$ are added to E_2 (and thus constitute $E_\cap$) for each $u \in \{v, w\}$. Since both stages are planar, they are also both weakly bipartite.

Let $M \geq 4$ be a constant. Both weight functions w_i, $i \in [2]$, assign weight 1 to edges in $E_\cap$ and weight M to edges in $E_i \setminus E_\cap$.

For a maximum cut in G_1, each of the $|V|$ vertex gadgets can be viewed independently. In the gadget for vertex v there are four vertex selections $S_1 \subseteq V_1$ that yield a maximum cut of weight $9M + 3$. They arise from two independent choices: (i) on the 6-cycle of M-weighted edges, pick either the three a-vertices or the three c-vertices into S_1 and (ii) pick either x_v or the three b-vertices into S_1. Note that, regardless of the vertex selections, for every maximum cut in G_1 the following statement holds true: in each vertex gadget either the ab-edges or the bc-edges constitute precisely $\delta(S_1) \cap E_\cap$.

For a maximum cut in G_2, each of the $|E|$ edge gadgets can be viewed independently. In the gadget for edge $e = vw$ there are four vertex selections S_2 that yield a maximum cut of weight $9M + 2$. They arise from two independent choices: (i) on the 8-cycle of M-weighted edges, pick every second vertex thus containing either the two a-vertices or the two c-vertices and (ii) pick one of the two b-vertices into S_2. Note that, regardless of the vertex selections, for every maximum cut in G_2 the following statement holds true: in each edge gadget exactly one ab-edge and one bc-edge are in $\delta(S_2)$.

Claim. *$\mathcal{J}$ is a yes-instance if and only if $\mathcal{I}$ is a yes-instance.*

Proof of Claim. We prove the two implications separately:

"$\Leftarrow$" Suppose there is an $S \subseteq V$ such that $|\delta(S)| \geq k$. We construct a multistage cut (F_1, F_2) by choosing a vertex selection $S_i \subseteq V_i$ for $i \in [2]$ and

setting $F_i := \delta(S_i)$. In G_1, pick all a-vertices into S_1. If $v \in S$, pick x_v into S_1; if $v \notin S$, pick the three $b^{\cdot}_v$-vertices into S_1. Thus, if $v \in S$, we have its ab-edges in F_1; otherwise its bc-edges. In G_2, for each edge gadget for edge $e = vw$, pick every second vertex of the 8-cycle into S_2 including the two $a^e_{\cdot}$-vertices. For each $e = vw \in \delta(S)$, assume w.l.o.g. $v \in S$ and $w \notin S$, and pick b^e_w into S_2. Thus, $a^e_v b^e_v$, $b^e_w c^e_w \in F_2$ and $a^e_w b^e_w$, $b^e_v c^e_v \notin F_2$. If $e \notin \delta(S)$, pick arbitrarily one of the two $b^e_{\cdot}$-vertices. Thus, $|\{a^e_v b^e_v, a^e_w b^e_w\} \cap F_2| = 1$ and $|\{b^e_v c^e_v, b^e_w c^e_w\} \cap F_2| = 1$.

Since the number of common edges in F_1, F_2 is optimized, in the edge gadget of each $e = vw \in E$, the number of edges in $F_1 \cap F_2$ is 2 if $e \in \delta(S)$ and 1 otherwise. Overall, this yields an intersection profit of $q_\cap(F_1, F_2) \geq |E| + k = \kappa$.

"⇒" Let (F_1, F_2) be a $q_\cap$-optimal multistage solution with $q_\cap(F_1, F_2) \geq |E| + k$. Since F_1 is a maximum cut in G_1, in each vertex gadget either all ab-edges or all bc-edges are in F_1. We construct a vertex selection $S \subseteq V$ in the original graph according to the following rule: pick v into S if and only if the ab-edges in the vertex gadget of v are in F_1.

Consider some $e = vw \in \delta(S)$. W.l.o.g., assume $v \in S$ and $w \notin S$. By construction, $a^e_v b^e_v$ and $b^e_w c^e_w$ are in F_1. Since F_2 is a maximum cut in G_2, in each edge gadget there is exactly one ab-edge and one bc-edge in F_2. Since F_2 is chosen such that $q_\cap(F_1, F_2)$ is maximum and each edge gadget in G_2 can be resolved independent from the other edge gadgets, $a^e_v b^e_v$ and $b^e_w c^e_w$ will be in $F_1 \cap F_2$. However, if $e \notin \delta(S)$, it is easily checked that in the corresponding edge gadget only a single edge can be in $F_1 \cap F_2$. By summing up $F_1 \cap F_2$ over all edge gadgets, we have $\kappa \leq |F_1 \cap F_2| = 2 \cdot |\delta(S)| + |E \setminus \delta(S)| = |E| + |\delta(S)|$ and thus $|\delta(S)| \geq k$. ◁

This concludes the proof. □

Theorem 40. *There is a $1/\sqrt{8\chi}$-approximation algorithm for* MWBMaxCut *and a $1/\sqrt{2\chi}$-approximation for* MWBMaxCut$|_2$.

Proof. Follow the proof of Theorem 31 but using $\varepsilon := -1/(|E| + 1)$ and, e.g., the algorithm by Grötschel and Pulleyblank [GP81] to compute a maximum cut in polynomial time on weakly bipartite graphs. □

8.5 MULTISTAGE MINIMUM WEIGHT BIPARTITE VERTEX COVER

The Vertex Cover problem is our first example of an SP that asks for a selection of vertices, equipped with individual weights. Since it is NP-hard in general [GJ79], we restrict our multistage version of the problem to bipartite stages. In contrast to the edge selection problems we have seen before, proving preficiency now requires a new proof.

In a bipartite graph $G = (V, E)$ with vertex weights $w\colon V \to \mathbb{N}_{>0}$, a vertex set $U \subseteq V$ is a *vertex cover* if each $e \in E$ is incident with some vertex in U. A vertex cover U has *minimum weight* if for each vertex cover U' we have $w(U) \leq w(U')$. MMinBVC aims to maximize the number of common vertices:

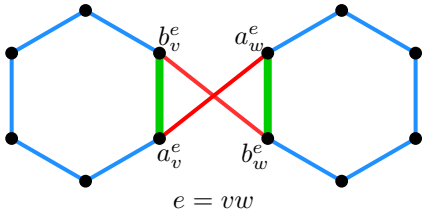


Figure 8.3: Two vertex gadgets and one edge gadget as used in the proof of Theorem 42: Bold green edges are in $E_\cap$, blue edges in $E_1 \setminus E_\cap$, red edges in $E_2 \setminus E_\cap$.

Definition 41 (MMINBVC). *Given a multistage graph $(G_i)_{i\in[\tau]}$ with vertex weight functions $w_i \colon V_i \to \mathbb{N}_{>0}$ for each $i \in [\tau]$ where each stage is bipartite, find a $q_\cap$-optimal multistage solution $(U_i)_{i\in[\tau]}$ such that for each $i \in [\tau]$, $U_i \subseteq V_i$ is a minimum weight vertex cover in G_i.*

Theorem 42. MMINBVC$|_2$ *is NP-hard already with uniform weights on multistage graphs where each stage only consists of disjoint cycles.*

Proof. We will perform a reduction from the unweighted MAXIMUM CUT problem on graphs with maximum degree 3 [Yan78] to the (unweighted) decision variant of MMINBVC: "Given a two-stage graph $\mathcal{G}$ where each stage is bipartite and a number $\kappa \in \mathbb{N}$, is there a $q_\cap$-optimal multistage solution (U_1, U_2) for MMINBVC with intersection profit $q_\cap(U_1, U_2) \geq \kappa$?" In MAXIMUM CUT, one is given an undirected graph $G = (V, E)$ and a natural number k; the question is to decide whether there is an $S \subseteq V$ such that $|\delta(S)| \geq k$.

Given an instance $\mathcal{I} := (G = (V, E), k)$ of MAXIMUM CUT, we construct an equivalent instance $\mathcal{J} := (\mathcal{G}, \kappa)$ of MMINBVC. Set $\kappa := |E| + k$. We start with an empty 2-stage graph $\mathcal{G} := (G_1, G_2)$. In G_1, we will have disjoint gadgets for each vertex, allowing three incident edges each. In G_2, we will have disjoint gadgets for each edge, intersecting with the two corresponding vertex gadgets in G_1 (cf. Figure 8.3). A minimum vertex cover in G_1 will correspond to a vertex selection in G and a minimum vertex cover in G_2 will allow us to count the edges of G that are incident to exactly one selected vertex.

In G_1, for each $v \in V$ we create a 6-cycle C_v and label its vertices counter-clockwise with a_v^e, b_v^e, $a_v^{e'}$, $b_v^{e'}$, $a_v^{e''}$, $b_v^{e''}$, where $\{e, e', e''\} = \delta(v)$ denote the edges incident with v (if $|\delta(v)| < 3$, the vertices are labeled with fictional indices). In G_2, we use the same vertex set and for each $e = vw \in E$ we create the 4-cycle C_e by introducing the edges $a_v^e a_w^e$, $a_w^e b_w^e$, $b_w^e b_v^e$, $b_v^e a_v^e$.

For a minimum vertex cover U_1 in G_1, for each C_v either all a-vertices or all b-vertices are in U_1. For a minimum vertex cover U_2 in G_2, for each C_e with $e = vw$, either $a_v^e, b_w^e \in U_1$ or $a_w^e, b_v^e \in U_1$.

Claim. *$\mathcal{J}$ is a yes-instance if and only if $\mathcal{I}$ is a yes-instance.*

Proof of Claim. We prove the two implications separately:

"$\Leftarrow$" Suppose there is an $S \subseteq V$ such that $|\delta(S)| \geq k$. We construct a vertex cover U_1 for G_1 as follows: for each $v \in S$, pick all a-vertices of C_v into U_1;

for each $v \in V \setminus S$, pick all b-vertices of C_v into U_1. Similarly, we construct a vertex cover U_2 for G_2: for each $e = vw$ with $v \in S$ and $w \notin S$ (i.e., $e \in \delta(S)$), pick a_v^e and b_w^e into U_2; for each $e \notin \delta(S)$, pick any two opposite vertices in C_e.

Consider some edge $e = vw \in E$. If $e \in \delta(S)$, $U_1 \cap C_e = U_2 \cap C_e$ and two vertices are chosen identically. If $e \notin \delta(S)$, $U_1 \cap C_v$ and $U_1 \cap C_w$ contain either both an a-vertex or both a b-vertex; since U_2 must contain one vertex of each type, exactly one vertex is chosen identically. Summing over all edge cycles yields $q_\cap(U_1, U_2) = |E| + |\delta(S)| \geq \kappa$.

"⇒" Let (U_1, U_2) be a $q_\cap$-optimal multistage solution with $q_\cap(U_1, U_2) \geq |E| + k$. We construct a vertex selection $S \subseteq V$ in the original graph according to the following rule: pick v into S if and only if the a-vertices of C_v are in U_1.

Consider some edge $e = vw \in E$. If $v \in S$ and $w \notin S$ (i.e., $e \in \delta(S)$), $U_1 \cap C_e = \{a_v^e, b_w^e\}$. Since U_2 maximizes the intersection with U_1 and can be chosen independently on each C_e, U_2 must be identical to U_1 on C_e; thus, $|U_1 \cap U_2 \cap C_e| = 2$. If $e \notin \delta(S)$, every minimum vertex cover U_2 can contain at most one vertex of $U_1 \cap C_e$. Summing up $U_1 \cap U_2$ over all edge gadgets, we have $\kappa \leq |U_1 \cap U_2| = 2 \cdot |\delta(S)| + |E \setminus \delta(S)|$ and thus $|\delta(S)| \geq k$. ◁

This concludes the proof. □

Theorem 43. *There is a* $1/\sqrt{8\chi}$*-approximation algorithm for* MMinBVC *and a* $1/\sqrt{2\chi}$*-approximation for* MMinBVC$|_2$.

Proof. We only need to show preficiency for MMinBVC and apply Theorems 25 and 27. Let $G = \big(V = (A, B), E\big)$ be a bipartite graph with vertex weights $w\colon V \to \mathbb{N}_{>0}$ and $Y \subseteq V$ the set of vertices to be preferred. Let $\varepsilon := 1/(|V| + 1)$. We construct the modified graph G' from G by adding two new vertices s, t and edge sets $\{sv \mid v \in A\}$ and $\{vt \mid v \in B\}$. We then equip G' with edge weights $w'(uv) := w(v) - \varepsilon \cdot \mathbb{1}\big(v \in Y\big)$ for all $u \in \{s, t\}$, and $w'(uv) := \infty$ otherwise.

Note that w' is well-behaved w.r.t. w. It is well-known that a minimum weight s-t-cut $C \subseteq E$ in G' (computable in polynomial time [FF56]) induces a minimum weight vertex cover U in G by picking all vertices $v \in V$ that are incident with an edge in C. Further, U maximizes $|U \cap Y|$: Suppose there is a minimum weight vertex cover U' in G with $|U' \cap Y| > |U \cap Y|$. Let C' again denote the s-t-cut associated with U'. By construction and since $w(U') = w(U)$, we have $w'(C') < w'(C)$, again contradicting minimality of C w.r.t. w'. □

8.6 MULTISTAGE MAXIMUM WEIGHT BIPARTITE INDEPENDENT SET

The last example is again based on a weighted vertex selection SP that is NP-hard on general graphs, Independent Set [GJ79]. While it has a maximization goal, we make use of its relation to Vertex Cover on bipartite graphs.

In a graph $G = (V, E)$ with vertex weights $w\colon V \to \mathbb{N}_{>0}$, a vertex set $U \subseteq V$ is an *independent set* if for $u, v \in U$ with $u \neq v$ we have $uv \notin E$. An independent set U has *maximum weight* if for each independent set U' we have $w(U) \geq w(U')$.

Definition 44 (MMAXBIS). *Given a multistage graph $(G_i)_{i\in[\tau]}$ with vertex weights $w_i\colon V_i \to \mathbb{N}_{>0}$ for each $i \in [\tau]$ where each stage is bipartite, find a $q_\cap$-optimal multistage solution $(U_i)_{i\in[\tau]}$ such that for each $i \in [\tau]$, $U_i \subseteq V_i$ is a maximum weight independent set in G_i.*

It is well-known that the complement of a minimum weight vertex cover is a maximum weight independent set. However, the complement of an optimal multistage vertex cover in general does not yield an optimal multistage independent set. Nonetheless, the former property is still key to the following proof.

Theorem 45. MMAXBIS$|_2$ *is NP-hard already with uniform weights on multistage graphs where each stage only consists of disjoint cycles.*

Proof. Follow the proof for Theorem 42. Equivalently to before, every cycle in every stage allows exactly two (sub)solutions. In particular, these are the very same subsolutions as before, since here maximum independent sets are identical to minimum vertex covers. To solve MAXIMUM CUT, one has to pick subsolutions with maximum intersection. □

Theorem 46. *There is a $1/\sqrt{8\chi}$-approximation algorithm for* MMAXBIS *and a $1/\sqrt{2\chi}$-approximation for* MMAXBIS$|_2$.

Proof. Follow the proof of Theorem 43, but using $\varepsilon := -1/(|V| + 1)$. Now, selecting the complement of the vertex cover yields an independent set with the maximum number of vertices from Y. □

9

SUMMARY

In this part, we considered multistage generalizations of Subgraph Problems that require an optimal solution in each individual stage while the transition quality is to be optimized. We provided two framework approximation algorithms for such MSPs: Algorithm 24 allows to generalize any 2-stage algorithm for any MSP to an unrestricted number of stages; Algorithm 28 is a 2-stage approximation algorithm for any preficient $q_{\cap}$-MSP. We then showcased the ease-of-use of our results by applying them to several natural MSP variants of well-known classical graph problems.

It remains open whether these algorithms are best possible polynomial-time approximations for any of the considered MSPs. In fact, there *cannot* be a general result establishing tightness for the whole class of MSPs as some problems are actually polynomial-time solvable (see, e.g., the vertex variants of MMinCut above).

For ease of exposition, we have only considered multistage generalizations of *subgraph* problems in this part. However, our techniques are also applicable to more general multistage *subset* problems, i.e., without the need of an underlying graph. This can be easily understood as all our proofs solely work on a set system on a ground set X. Alas, we know of no natural multistage non-subgraph optimization problem that simultaneously is (a) NP-hard, (b) preficient, and (c) not trivially reformulated as an MSP.

Part III

EXPERIMENTAL STUDY: MULTISTAGE SHORTEST PATH

The long and winding road
That leads to your door
Will never disappear
I've seen that road before
— The Beatles: *The Long and Winding Road* [Bea70]

10

BACKGROUND

This part is based on joint work with Markus Chimani that has been published in the conference proceedings of the *2nd Symposium on Algorithmic Foundations of Dynamic Networks* (SAND 2023) [CT23].

There are several theoretical results on different multistage problems and their complexities, as well as FPT and approximation algorithms. However, there is a severe lack of experimental validation and resulting feedback. Not only are there no such algorithmic experiments (except for a recent comparison of heuristics for a niche subset problem [BCE+23]), we do not even know of any strong set of multistage benchmark instances.

In this part we want to improve on this situation. We consider the Multistage Shortest Path (MSPath) problem. MSPath was first proposed in [GTW14] and introduced with a trade-off objective in [FNS+20]. We discuss it here in the setting where we only allow optimal solutions per stage: Given an edge-weighted multistage graph and a vertex pair (s, t), find a shortest s-t-path in each stage such that the subsequent paths are as similar as possible (see Section 10.2 for a formal definition). For example, in a transportation scenario, it might be necessary but expensive to prepare each road segment before using it. Thus, we want a collection of shortest paths that allows us to reuse as many segments as possible. In a communication scenario, we prefer to use recently established channels, but not at the cost of sacrificing transfer speed. If the optimality requirement per stage appears too restricting, we point out that one may potentially relax it in practice by altering the notion of what a *shortest* path is, e.g., by rounding edge weights so that all paths of reasonably similar length are considered optimal.

10.1 CONTRIBUTION

In this part we improve on the state-of-the-art regarding practical algorithmics in the following two ways: First (Chapter 11), we propose the first rich benchmark sets for a multistage graph problem, ranging from synthetic to real-world data. We take special care to avoid ad-hoc generation schemes and parameterizations which, in case of MSPath, would typically only yield rather trivial instances.

Instead, we devise several explicit measures of reasons for triviality and actively seek schemes and parameterizations to avoid them. Secondly, we implemented and tested a set of heuristic, approximate, and exact algorithms to tackle MSPath in practice (Chapter 12), and we report on our exploratory study (Chapter 13). A focus of this study is to test the consistency between theoretical results and their practical realization and use it as a source for identifying new research questions.

Theoretical research suggests to improve on algorithms for the (formally already hard) 2-stage problem variant MSPath$|_2$, as we only know a single approximation algorithm (with non-constant approximation ratio). The multistage variants with more than two stages can reuse any 2-stage algorithm while weakening its approximation ratio only by the constant ratio of $1/2$. Interestingly, we find that in practice solutions for MSPath$|_2$ can be approximated (and even computed exactly) rather simply, but neither the known approximation nor other heuristics yield satisfactory results for general MSPath. Thus, we propose that a promising step for theoretical research would be to further investigate the intricacies of the true multistage setting instead of relying on algorithms for a small constant number of stages.

The implementations will be part of the open-source (GPL) *Open Graph algorithms and Data structures Framework* [CGJ+13] (`http://www.ogdf.net`); all benchmark sets and experimental data are available at `https://tcs.uos.de/research/msp`.

10.2 MULTISTAGE SHORTEST PATH

Given a graph G with positive edge weights $w\colon E(G) \to \mathbb{Q}^+$, we encode a path $P \subseteq E(G)$ as an edge set and denote its *path length* by $\ell(P) := \sum_{e\in P} w(e)$. Given a *query* $(s,t) \in V(G)^2$, a *shortest s-t-path* is an s-t-path with minimum path length. In contrast, we may also consider the number of *hops* (edges) $|P|$ of a path P. The *hop-distance* $h(s,t)$ is the smallest number of hops over all s-t-paths.

Definition 47 (MSPath)**.** *Given a multistage graph $(G_i)_{i\in[\tau]}$, positive edge weights $w_i\colon E_i \to \mathbb{Q}^+$ for each $i \in [\tau]$, and a query $(s,t) \in V^2$, find a sequence $(P_i)_{i\in[\tau]}$ of paths such that each P_i is a shortest s-t-path in G_i and the transition quality $Q(\mathcal{P}) := \sum_{i\in[\tau-1]} |P_i \cap P_{i+1}|$ is maximized.*

If there is an upper bound T on the number of stages τ, MSPath *may be denoted by* MSPath$|_T$.

While the classical shortest s-t-path problem is long known to be efficiently solvable using Dijkstra's algorithm [Dij59], MSPath$|_2$ was shown to be NP-hard even for unweighted instances via a reduction from 3Sat [FNS+20]. Although not stated explicitly, the proof can easily be adapted to an approximation-preserving reduction from Max-2Sat showing that, unless P = NP, MSPath$|_2$ does not admit a PTAS nor a constant-factor approximation with factor better than $21/22$ [Hås97]. In [FNS+20], several similarity and dissimilarity measures

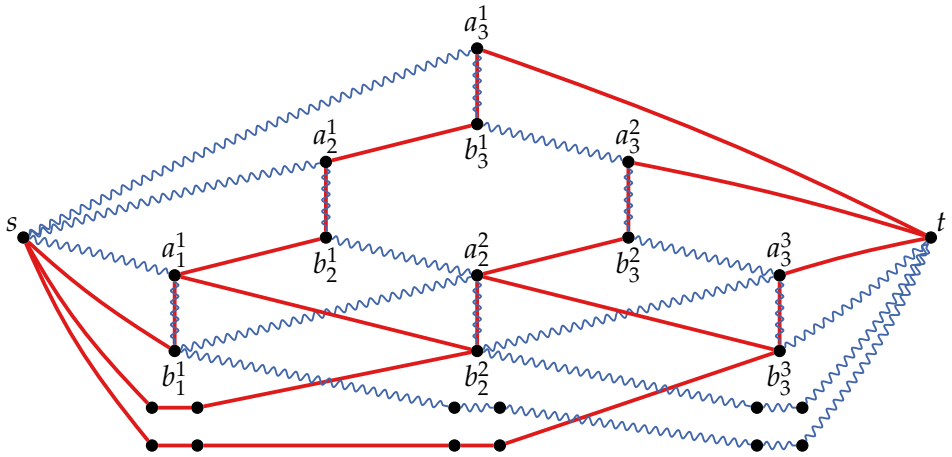


Figure 10.1: MSPATH$|_2$ instance with $k = 3$ as in Lemma 48. Edges in E_1 are curvy and blue, edges in E_2 straight and red.

were considered, and several results w.r.t. the parameterized complexity of MSPATH could be established.

As the problem is NP-hard, we may be interested in approximate solutions. The only known approximation algorithms for MSPATH$|_2$ and general MSPATH arise as special cases of the general approximation framework presented in Part II. We will briefly summarize the adapted algorithms in Sections 12.2 and 12.3. For MSPATH and MSPATH$|_2$, Algorithm 24 and Algorithm 28 yield approximation ratios of $1/\sqrt{8\chi}$ and $1/\sqrt{2\chi}$, respectively. While Theorems 25 and 27 guarantee these ratios, their tightness cannot be deduced for arbitrary subgraph problems. However, following the construction ideas of Lemma 10 from Part I, we can show the tightness of the ratio (up to a small constant) for MSPATH$|_2$, which also implies the respective tightness of the MSPATH-algorithm:

Lemma 48. *The approximation ratio of Algorithm 28 for* MSPATH$|_2$ *is at most* $1/\sqrt{2\chi}$.

Proof. We construct a family $(\mathcal{G}_k, s_k, t_k)_{k\in\mathbb{N}}$ of MSPATH$|_2$ instances, parameterized by some $k \geq 2$. An example using $k = 3$ is depicted in Figure 10.1.

In the construction description, we denote $s_k = s$ and $t_k = t$ for simplicity. In the first stage, we have k disjoint s-t-paths $P_1, \ldots, P_k$ of length $2k + 1$; the inner vertices of path P_i with $i \in [k]$ are labeled $a_i^1, b_i^1, a_i^2, b_i^2, \ldots, a_i^k, b_i^k$. We add an additional s-t-path P_* of length $2k + 1$ by adding edges $b_i^i a_{i+1}^{i+1}$ for each $i \in [k-1]$.

In the second stage, we again construct k disjoint s-t-paths $P^1, \ldots, P^k$ of length $2k + 1$, reusing some ab-edges in reversed direction. For each $i \in [k]$, path P^i is the s-t-path from s through $b_i^i, a_i^i, b_{i+1}^i, a_{i+1}^i, \ldots b_k^i, a_k^i$ to t; the first edge sb_i^i is subdivided into $2i - 1$ edges to obtain paths of equal length. We create an additional s-t-path P^* of length $2k + 1$ by adding edges $a_i^i b_{i+1}^{i+1}$.

In each stage, exactly the named paths are shortest s-t-paths. The only edges common to both stages are $E_\cap = \{a_i^j b_i^j \mid i \in [k], j \in [i]\}$ and thus $\chi = k(k+1)/2$. The optimal MSPATH$|_2$ solution is (P_*, P^*) containing $k = (\sqrt{8\chi + 1} - 1)/2$ common edges; any other solution contains at most one common edge.

Algorithm 49: Preprocessing forbidden elements in an edge-weighted graph $G = (V, E)$

1 compute shortest path distances $d(v)$ from s to each $v \in V$ using Dijkstra's algorithm
2 remove all edges $\{uv \in E \mid d(u) + w(uv) \neq d(v)\}$
3 compute all vertices U with a path to t (via breadth-first search from t with reversed edges)
4 remove all vertices $V \setminus U$

Algorithm 28 may never choose the optimal P_* as the s-t-path for the first stage: In the first iteration, both P_* and P_k contain k edges of $E_\cap$, so the algorithm may choose P_k and obtain a MSPath$|_2$ solution with intersection quality 1. In the following iteration, the weight (denoting the preference of edges) of P_* has decreased by 1, since the edge $a_k^k b_k^k$ has already been chosen in P_k. Thus, in each iteration $i \in [k]$, the algorithm may choose P_{k-i+1} over P_*, decreasing the weight of P_* to $k - i$. After choosing P_1 in iteration k, each edge of $E_\cap$ has been in some shortest s-t-path in the first stage; the algorithm stops and returns a MSPath$|_2$ solution with intersection quality 1. The approximation ratio of Algorithm 28 is thus at most $1/k = 2/(\sqrt{8\chi + 1} - 1)$, which tends to $1/\sqrt{2\chi}$ for increasing k. □

10.3 PREPROCESSING

For a given query $(s, t) \in V^2$ and looking at any stage individually, we may remove all its *forbidden* edges (i.e., edges that are not in any shortest s-t-path in that stage) without altering the set of feasible solutions. We may also discard (arising) vertices of degree 0. This can be done efficiently, as given in Algorithm 49. Thus, in the following we will assume that this preprocessing is always performed before running the actual algorithms. A *reduced* stage G_i, i.e., a stage that has been preprocessed w.r.t. the instance's query (s, t), has the following useful properties:

1. G_i is a directed acyclic graph (*DAG*), and
2. for each vertex $v \in V(G_i)$, all paths from s to v have the same length. The same holds for all paths from v to t.

After the stage-wise preprocessing, both properties in particular also hold for the graph induced by the intersection $E_i \cap E_{i+1}$, for each $i \in [\tau - 1]$.

11

BENCHMARK INSTANCES

Multistage problems have mostly been viewed from a theoretical perspective up to now, and there are thus no established sets of *stage-wise* temporal instances available. Furthermore, it turns out that acquiring and even generating reasonable instances is no easy feat: In our investigations, we learned that most ad-hoc generation schemes typically lead to rather trivial multistage instances. If there are only very few different (or even just one unique) shortest paths per stage, there is not much room for transition optimization; if there are several shortest paths but on very similar stages, chances are that a single solution path can be chosen throughout all stages; if the stages become too dissimilar, such that they have only few edges in common between shortest paths, it again becomes rather simple to select shortest paths that agree in terms of these edges between subsequent stages.

An adversary may argue that such issues would go away if one switches to a trade-off based objective function where the paths' lengths are allowed to deviate from the optimum in order to allow better transitions. But we disagree: Generating instances with only a trade-off based objective function in mind would easily hide the fact that such instances may become trivial for different balancing ratios between the two considered objective functions. If, however, the instances are well-designed for our scenario with truly optimal shortest paths, we expect them to be also interesting for trade-off based optimization. Thus, it is important to discuss our benchmark generating procedure in more detail than is often done otherwise. While we cannot guarantee that our instances are especially sensible for problems beyond MSPath, we hope that the underlying generation methods and considerations may be useful for devising new instances for experimental studies on other multistage problems as well.

We consider four different types of benchmark instances, each with slightly different focus and motivation (see Section 11.1), and ranging from highly synthetic to real-world origins. After discussing schemes of obtaining MSPath instances from underlying graphs in Section 11.2, we discuss the complexities of identifying good parameters to obtain non-trivial MSPath instances in Section 11.3. Then, Section 11.4 presents the final technical parameterizations of our benchmark sets and resulting instance properties.

11.1 RATIONALE FOR THE BENCHMARK SCENARIOS

As discussed in Section 10.3, the query-specific preprocessing of MSPath instances may yield vastly smaller stages, and the reduced stages have a very specific structure. In particular, their size is not necessarily related to the original instance size anymore. To obtain interesting instances, a main goal is thus to

generate instances with a reasonable number of shortest paths with a reasonable number of hops each, so that the preprocessing does not already essentially solve the instance.

To this end, we start with generating a highly synthetic benchmark set `grid`, which consists of long grid graphs (i.e., two-dimensional grids where one dimension is significantly smaller than the other). For a query (s, t) where s (t) is the lower left (upper right, respectively) corner of the grid, these graphs (assuming unit edge weights) already resemble reduced MSPath instances. Further, in contrast to more quadratically-shaped grid graphs, even relatively small modifications to the graphs are likely to yield non-trivial instances.

The benchmark set `geom` contains nearest-neighbor graphs [EPY97], generated by a random point set in the Euclidean plane. Such random graphs allow for multiple shortest paths of reasonable lengths. In contrast, other well-established randomized generation paradigms like Erdős-Renyi graphs or Barabási-Albert graphs would only yield very small diameters [CL01; BR04]. Our geometric graphs arguably have naturally occurring edge weights. Further, if one stage is generated from the previous stage by adding some random displacement to each vertex, they also provide a natural temporal relationship between consecutive stages.

The probably most natural application for shortest path queries is navigation in road networks. However, readily available data sets do not include temporal data suitable for MSPath. Our benchmark set `hybr` thus uses real-world road networks as the underlying graph data, for which we artificially generate temporal differences between the stages. Our modification methods (see Section 11.2) are mainly motivated by this scenario, but we use the same modification methods for the previous two benchmark sets as well.

Finally, there exist real-world data sets from other applications that include time-stamped edges. Under those, we are mainly interested in email communication networks ("who wrote to whom, and when?") or human contact networks ("who was near whom, and when?"), as we can interpret these data in the context of the MSPath problem: We want to quickly pass some information from source to target, while preferring interpersonal relations that have been used recently. We collect such instances in our benchmark set `real`.

11.2 MULTISTAGE INSTANCE GENERATION

MODIFICATION VARIANTS. Necessarily, the stages of a multistage graph need to differ to compose a non-trivial instance. The `real` instances already have differing stages; for `grid`, `geom`, and `hybr` base instances we can use either of the following three modification schemes (applied to each stage independently) to obtain multiple differing stages. Additionally, we can also obtain differing stages for the `geom` instances by perturbing the vertex coordinates between stages to simulate random walks of the vertices (see Section 11.4). Keep in mind that these modifications are performed on the original graph, prior any query knowledge and thus prior to any preprocessing.

Edge deletion: In most real-world interpretations of multistage graphs, there are reasons for the absence of some edges in some stages (as otherwise the multistage framework would probably not be the best model). For example, road closures in road networks or link failures in computer networks can lead to their temporary unavailability. Given a *modification ratio* $\lambda_E \in [0, 1)$, we remove $\lfloor \lambda_E \cdot |E_i| \rfloor$ many edges, chosen uniformly at random, from each G_i.

Vertex deletion: Removing arbitrarily chosen edges might (i) not alter the set of shortest paths too much and (ii) not describe all real-world scenarios too well, as the reason for the absence of an edge can have some local impact on surrounding edges as well. A simple method to generate local events of slightly larger impact is to remove vertices together with all incident edges—this occurs, e.g., if some road intersection is blocked, or if some server goes offline in a communication network. As above, we use a *modification ratio* $\lambda_V \in [0, 1)$ and remove $\lfloor \lambda_V \cdot |V| \rfloor$ many vertices, chosen uniformly at random, from each G_i.

Weight scaling: Some incidents (e.g., construction sites) do not render the respective vertex or edge completely unusable but rather increase the cost for their usage. This is typically no isolated effect but also affects the neighborhood of said graph element—the closer the proximity, the larger the effect. We model this by selecting a random vertex v and sorting E_i by hop-distance from v (i.e., the minimum of the hop-distances between v and the edge's endpoints); the closer an edge is to v, the more we scale up its weight. The following precise parameters were selected subject to the discussion in Section 11.3: the weights of the $\lfloor |E_i|/8 \rfloor$ closest edges are multiplied by a factor of 4; the next $\lfloor |E_i|/4 \rfloor$ edges are multiplied by a factor of 2. Observe that if edges have exponential weights (see details of *Random Nearest Neighbor Graphs* in Section 11.4) they retain this property after the scaling.

QUERY SELECTION. While the `grid` instances are constructed with specific extremal queries in mind, we need to choose queries for the other three benchmark sets. To find multiple queries, each with relatively long shortest paths, we use the following randomized process. Consider some stage G_i with the least number of edges. Let $h'(v, w) := h(v, w)$ denote the hop-distance from v to w in G_i if they are in a common component, and $h'(v, w) := -1$ otherwise. Let $h^*(v) := \max_{u \in V} h'(v, u)$ and $H(v) := \{w \in V \mid h'(v, w) \geq 3/4 \cdot h^*(v)\}$ denote a set of distant vertices from v. Starting from a random vertex $c \in V$ in the largest connected component of G_i, we choose the source s uniformly at random from $H(c)$. The target t is in turn chosen uniformly at random from $H(s)$. If the query is not feasible for all stages, it is rejected.

11.3 QUALITY CRITERIA AND TRIVIALITY CONSIDERATIONS

To differentiate interesting from trivial instances, multiple aspects have to align. These are specifically derived from our view on the MSPath-problem but might also be generalized to classify instances for other multistage problems.

TRIVIALITY IN A STAGE. Some trivialities can be pinpointed to a single stage.

Few paths: Let N denote the number of shortest s-t-paths in a single stage G_i. If $N = 0$, vertex t is not reachable from s and the instance could be split into two independent sub-instances before and after the ith stage. Alternatively, if a practical application would rather incentivize similarity to the last feasible solution, we could simply remove the infeasible stage. Similarly, if $N = 1$, the shortest s-t-path is unique in stage i, and we can split the instance at this stage. The case $N \leq 1$ is trivial to check after preprocessing, since then E_i is either empty or a single path. We may discard instances with such a stage for experimental purposes. As N may be exponential in $|E_i|$, for $N \geq 2$, we consider the ratio $|E_i|/h_i(s,t)$ as a measure to estimate the non-triviality of stage G_i. Here, h_i is the hop-distance in G_i, easily computable during preprocessing.

Short paths: We may disregard instances with too small h_i, as defined above.

TRIVIALITY IN A TRANSITION. Trivialities arising in transitions between stages may be harder to spot. Furthermore, even after understanding how to generate instances with reasonable stage-wise non-triviality, finding generation parameters that yield transition-wise non-trivial instances turned out to be much more fiddly and required more computational effort. E.g., to compute (or estimate) the measures below, we require optimal (or heuristic) solutions. Section 12.1 discusses a method to (in practice) acquire such an optimal solution $(P_i)_{i\in[\tau]}$ in reasonable enough time by the use of an ILP.

Small intersection: If, after preprocessing, the intersection $E' := E_i \cap E_{i+1}$ between two consecutive stages is too small or poorly structured (e.g., if E' consists of mostly disconnected edges), all of E' might be in an optimal solution simultaneously, i.e., $E_i \cap E_{i+1} \subseteq P_i \cap P_{i+1}$. In this case, already the simplest greedy algorithm (see Section 12.2) would always find an optimal solution. We introduce the triviality measure $t_S := \frac{|P_i \cap P_{i+1}|}{|E_i \cap E_{i+1}|}$ which compares the optimal transition quality to the intersection size. If $t_S = 1$, the transition is trivial; if t_S is close to 0, this triviality aspect plays no important role.

Large intersection: If, on the other hand, $E_i \cap E_{i+1}$ is too large, each solution edge may always also be an intersection edge, i.e., $P_i = P_i \cap E_{i+1}$ for any shortest path P_i in G_i (and similarly for P_{i+1}). We introduce the triviality measure $t_L := \frac{2\cdot|P_i \cap P_{i+1}|}{|P_i|+|P_{i+1}|}$, comparing the optimal transition quality with

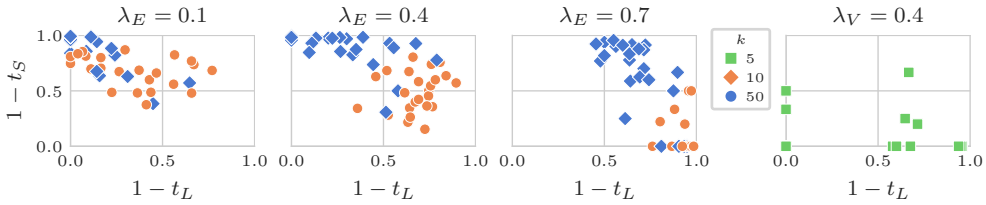


Figure 11.1: Triviality measures for 2-stage `geom` instances.

the mean number of hops of the respective shortest paths. If $t_L = 1$, the transition is trivial; if t_L is close to 0, this aspect is not important.

Small transition quality: Both triviality measures t_S and t_L are not very expressive if the optimal transition quality $|P_i \cap P_{i+1}|$ is low.

IDENTIFYING NON-TRIVIAL INSTANCES. The selection of the underlying graphs (and/or their generation methods) allows us to control the non-triviality within single stages in a reasonable and predictable manner. However, controlling the transition-based triviality (mainly t_S and t_L) turns out to be much more challenging. This is furthered by the fact that for a nice set of benchmarks, we would like to have similar parameterizations over all instance classes. To this end, we required multiple rounds of generating many instances starting with vastly diverse parameter selections until honing in with fine-grained parameter differences. While this may seem straight-forward on first sight, there are sometimes only very small ranges of suitable parameter values, and they may vastly drift or even disappear by slight changes to other parameters due to the interdependencies of the parameters.

We exemplarily discuss the effects on t_S and t_L by varying the modification parameters for `geom`. See Figure 11.1 for a visualization, where instances (points in the figure) that are trivial due to a too small (large) intersection tend to the horizontal (vertical, respectively) axis. Consider neighborhoods of size $k = 10$. For a small edge deletion modification ratio λ_E, the intersection is mostly too large, causing low $1 - t_L$; for larger λ_E, the point set moves down and to the right, rendering more instances to have low $1 - t_S$. This plausible effect is evident for all instance sets, albeit with large discrepancy for sensible values of λ_E depending on, for example, k: while $\lambda_E = 0.4$ is a sensible choice for $k = 10$, $k = 50$ would benefit from a higher λ_E value. Even more so, for some instance parameterizations (e.g., $k = 5$ and $\lambda_V = 0.4$), both t_S and t_L are likely to trigger. Thus, the selected parameters are a compromise between comparability of parameter values and non-triviality w.r.t. all of the above triviality considerations.

Table 11.1: **Instance characteristics**, grouped by relevant parameters. Here, n and m (n' and m') are the mean number of vertices and edges before (after, respectively) preprocessing, always understood as the union over all stages. The ratios n'/n and m'/m thus measure the effectiveness of the preprocessing strategy. h denotes the mean hop-count of a shortest path per stage; larger numbers typically indicate higher problem difficulty. Value χ gives the mean intertwinement of the considered instances after preprocessing.

grid	$y = 100$		$y = 200$		$y = 500$		$y = 1000$	
x	$\frac{n'}{n}$ / $\frac{m'}{m}$	χ / h	$\frac{n'}{n}$ / $\frac{m'}{m}$	χ / h	$\frac{n'}{n}$ / $\frac{m'}{m}$	χ / h	$\frac{n'}{n}$ / $\frac{m'}{m}$	χ / h
5	97.7%	362.8	98.0%	626.1	99.2%	1396.5	99.4%	2605.9
	97.1%	105.0	97.3%	208.6	98.6%	519.6	98.9%	1037.4
10	89.6%	676.5	87.8%	962.3	90.7%	1892.5	93.9%	3476.2
	88.0%	110.2	85.5%	214.8	87.7%	530.6	90.8%	1069.0
25	86.9%	2356.1	77.8%	3540.0	68.2%	4347.4	72.0%	6834.9
	86.0%	123.5	76.1%	227.0	64.8%	538.6	67.0%	1069.2
50	91.1%	5506.0	80.8%	9321.3	64.8%	14414.3	55.4%	13940.5
	90.7%	148.0	80.1%	249.0	63.0%	566.4	52.1%	1106.7

geom	$n = 1000$		$n = 2000$		$n = 5000$	
k	$\frac{n'}{n}$ / $\frac{m'}{m}$	χ / h	$\frac{n'}{n}$ / $\frac{m'}{m}$	χ / h	$\frac{n'}{n}$ / $\frac{m'}{m}$	χ / h
5	21.6%	53.6	17.1%	79.0	13.7%	141.4
	10.5%	30.8	8.4%	47.2	6.9%	74.1
10	20.5%	60.7	18.1%	100.6	15.0%	213.3
	7.7%	20.7	6.6%	30.9	5.7%	54.3
25	18.1%	120.6	17.2%	228.1	15.5%	505.0
	4.7%	12.4	4.5%	18.5	4.0%	35.7
50	13.1%	165.5	15.1%	335.2	15.6%	988.0
	2.3%	7.3	3.0%	13.3	3.2%	20.3

hybr	n	m	$\frac{n'}{n}$	$\frac{m'}{m}$	χ	h
CA	2.0M	2.8M	0.11%	0.09%	571.3	747.7
PA	1.1M	1.5M	0.19%	0.15%	551.4	653.6
TX	1.4M	1.9M	0.19%	0.15%	654.0	860.0

real	n	m	$\frac{n'}{n}$	$\frac{m'}{m}$	χ	h
dnc.24	401.8	1.2K	6.9%	5.8%	11.5	6.3
dnc.48	536.4	1.7K	5.5%	5.1%	15.0	5.4
enron.168	5.3K	12.4K	1.0%	0.8%	11.1	7.7
enron.744	8.9K	25.2K	0.5%	0.4%	12.4	7.4

11.4 FINAL PARAMETERIZATION AND GENERATION DETAILS

After multiple rounds of investigations to identify reasonable and consistent parameterizations that yield non-trivial instances, we finally arrive at the following generation settings. Table 11.1 shows key figures of the generated multistage instances, given as averages over the indicated instance classes. A detailed overview of the generation parameters is given in Tables 11.2–11.4 at the end of the section. The full set of benchmark instances and experimental results can be found at `https://tcs.uos.de/research/msp`.

`grid`: LONG GRID GRAPHS. The underlying grids have $|V| = x \cdot y$ for $(x, y) \in \{5, 10, 25, 50\} \times \{100, 200, 500, 1000\}$ and unit edge weights. For each of the three modification variants we generate MSPATH instances with 16 stages; each stage is derived from the original underlying graph. We consider the modification

ratios $\lambda_E \in \{1/5, 1/10, 1/20\}$ (for $x = 5$, $\lambda_E = 1/5$ is omitted due to generating mostly infeasible instances) and $\lambda_V = 1/20$. Overall, we generate 36 instances for each parameter combination, so we obtain 2736 `grid` instances.

`geom`: RANDOM NEAREST NEIGHBOR GRAPHS. We use several parameters to generate MSPATH instances with 16 stages. The vertices in G_1 are $n \in \{1000, 2000, 5000\}$ randomly chosen real-valued points uniformly distributed over the unit square $[0,1]^2$. We obtain the vertex position for each G_{i+1} from G_i by moving each vertex independently in a random direction chosen uniformly from $[-\frac{\varrho}{n}, \frac{\varrho}{n}]^2$ with $\varrho \in \{0, 1, 5\}$. In every stage, for each vertex we add an edge to its $k \in \{5, 10, 25, 50\}$ nearest neighbors (according to Euclidean distances). To allow for multiple shortest paths per stage, we consider two different weight functions: unit weights and exponential weights. The latter are generated by $2^{\lceil \log_2 100d \rceil}$, where d is the Euclidean distance. This mapping partitions the otherwise very diverse edge weights into buckets of (exponentially) similar weights. Due to the factor 100 (and that we have a nearest neighbor graph) we mostly observe weights in $\{1, 2, 4, 8, 16\}$.

We consider these graphs with the *edge deletion* (with $\lambda_E \in \{1/2, 1/20\}$, omitting $\lambda_E = 1/2$ for $k = 5$ and $\lambda_E = 1/20$ for $(k, n) = (50, 5000)$ due to triviality) and the *weight scaling* modifications. We do not use *vertex deletion* here, as these modifications (even for small non-trivial values of λ_V) resulted in mostly trivial instances (especially many infeasible ones). However, unless $\varrho = 0$, we also consider the instances without any further modifications since the random walks of the vertices already establish differences between the stages. Overall, we generate 4 instances for each of the 240 parameter combinations with a unique query as described in Section 11.2. We obtain 960 `geom` instances overall.

`hybr`: ROAD NETWORKS. We use the undirected variants of the popular real-world `roadNet` data set [LLD+08], namely the road networks of California (`CA`), Pennsylvania (`PA`), and Texas (`TX`) as three distinct underlying graphs. As the original data set does not contain any temporal information, we use the three modification variants to obtain multistage instances, with $\lambda_E \in \{1/10, 1/20, 1/100\}$ and $\lambda_V \in \{1/20, 1/100\}$. In contrast to the artificial graphs, we observe that preprocessing the underlying graph w.r.t. a query yields dramatically smaller graphs (see Table 11.1). Thus, we performed the stage-wise modifications *after* preprocessing (i.e., we first choose a random query as described in Section 11.2, then preprocess, modify and check feasibility), in order to guarantee that the modifications are significant within the stages. The benchmark set `hybr` consists of overall 360 multistage instances with 4 stages and a unique query each.

`real`: COMMUNICATION NETWORKS. Many real-world graph data sets with timestamped edges are email data sets [KY04; Kun13], where vertices represent people and an edge indicates a message exchange at the indicated times. We use the data sets as provided by the Konect graph collection [Kun13], but consider edges to be undirected. Here, timestamps are given with a relatively

high resolution ranging between 1 and 100 seconds, meaning that only very few events happen exactly at the same time. To generate stages with a non-trivial number of edges, we have to decrease the temporal resolution, i.e., we generate stages by accumulating all events that occur during some time window. If we choose too large time windows, the stages become too dense and yield only very short shortest paths. On the other hand, if we have too many stages, there are typically no feasible non-trivial queries possible. We thus pick time window sizes that yield interesting graphs, but restrict ourselves to 2 or 8 consecutive stages. The queries are selected as described in Section 11.2.

- `enron` [KY04; Kun13]: Email communication between employees of the energy corporation *Enron*. We use time window sizes of 168 hours (a week) and 744 hours (a month). To avoid too sparse (or obviously mislabeled) data, we only consider timestamps between May 27, 1998 and Feb 04, 2004 (a span of 297 weeks).
- `dnc-email` [Kun13]: Email communication between members of the US Democratic National Committee. We use time window sizes of 24 and 48 hours. Here, we consider timestamps between Sep 16, 2013 and May 25, 2016 (a span of 140 weeks).

The benchmark set `real` consists of 80 2-stage (66 × `enron`, 14 × `dnc-email`) and 20 8-stage instances (14 × `enron`, 6 × `dnc-email`) that are selected as those instances with the lowest triviality score, which is calculated as the sum over the values $10 \cdot \mathbb{1}\left(t_S = 1 \vee t_L = 1\right) + t_S \cdot t_L$ for the considered transitions.

We also conducted the same process on human contact data sets [ISB+11; EP06], where a timestamped edge indicates a measurement of physical proximity at the given point in time. However, the resulting graphs had either a very low diameter (3 or even lower, rendering MSPath essentially trivial) if the time windows were too wide, or were highly disconnected if the time windows were too narrow. There was no sweet spot between these effects and thus these instances are not included. We also note that research tells us that autonomous systems and similar networks typically experience shrinking diameters over time [LKF05], and are thus not well-suited to yield non-trivial MSPath instances.

Table 11.2: **Instance characteristics of `hybr` graphs.** Columns as in Table 11.1.

`hybr`	`CA`				`PA`				`TX`			
	$\frac{n'}{n}$	$\frac{m'}{m}$	χ	h	$\frac{n'}{n}$	$\frac{m'}{m}$	χ	h	$\frac{n'}{n}$	$\frac{m'}{m}$	χ	h
$\lambda_E = 0.01$	0.08%	0.06%	744.1	668.0	0.13%	0.10%	752.9	608.1	0.13%	0.10%	945.2	813.2
$\lambda_E = 0.05$	0.12%	0.09%	472.6	763.8	0.19%	0.15%	474.1	648.5	0.20%	0.16%	510.8	857.3
$\lambda_E = 0.1$	0.15%	0.12%	355.5	876.2	0.23%	0.19%	316.4	700.6	0.26%	0.20%	306.8	939.7
$\lambda_V = 0.01$	0.08%	0.06%	684.5	677.1	0.14%	0.11%	792.2	626.0	0.13%	0.10%	896.0	816.1
$\lambda_V = 0.05$	0.13%	0.11%	423.6	782.2	0.21%	0.17%	409.9	656.3	0.23%	0.18%	451.1	878.2
`scaling`	0.11%	0.08%	747.5	719.0	0.23%	0.18%	562.6	682.3	0.18%	0.14%	813.9	855.4

Table 11.3: **Instance characteristics of `grid` graphs.** Instance sets with "—" are not generated, while "⁄" indicates no known optimal solution. A mean hop-count value in italics is ignoring instances with no known optimum.

`grid`	$x = 5$				$x = 10$				$x = 25$				$x = 50$			
	$\frac{n'}{n}$	$\frac{m'}{m}$	χ	h	$\frac{n'}{n}$	$\frac{m'}{m}$	χ	h	$\frac{n'}{n}$	$\frac{m'}{m}$	χ	h	$\frac{n'}{n}$	$\frac{m'}{m}$	χ	h
$n = 100, \lambda_E = 0.05$	96.0%	95.3%	349.8	104.5	91.4%	90.6%	911.9	108.1	96.3%	96.1%	3531.4	*123.0*	97.8%	97.7%	7771.4	⁄
$n = 100, \lambda_E = 0.1$	97.7%	96.8%	245.0	108.2	81.5%	79.4%	395.9	109.8	83.1%	82.3%	2206.0	*123.0*	90.5%	90.1%	5802.2	⁄
$n = 100, \lambda_E = 0.2$	—	—	—	—	84.6%	80.2%	198.4	117.1	60.6%	57.4%	455.1	124.5	69.7%	68.0%	2258.6	*148.0*
$n = 100, \lambda_V = 0.05$	97.2%	96.5%	340.5	104.5	90.6%	89.8%	824.5	108.1	95.5%	95.3%	3204.2	*123.0*	97.6%	97.6%	7029.1	⁄
$n = 100$, `scaling`	99.9%	99.8%	515.8	103.0	100.0%	100.0%	1051.7	108.0	99.2%	99.1%	2383.9	123.0	100.0%	100.0%	4668.6	148.0
$n = 200, \lambda_E = 0.05$	97.1%	96.3%	587.7	207.8	82.3%	80.6%	1003.3	209.4	83.3%	82.7%	5291.2	⁄	92.4%	92.2%	14171.8	⁄
$n = 200, \lambda_E = 0.1$	98.3%	97.2%	405.3	215.5	83.9%	80.7%	551.4	215.3	60.3%	58.5%	1885.6	*223.7*	75.4%	74.7%	8791.3	⁄
$n = 200, \lambda_E = 0.2$	—	—	—	—	89.3%	84.2%	306.9	231.7	60.4%	54.8%	449.1	234.3	44.6%	42.2%	1272.7	249.8
$n = 200, \lambda_V = 0.05$	96.5%	95.4%	502.5	208.1	83.9%	82.1%	871.8	209.7	84.8%	84.3%	4763.4	*223.0*	91.8%	91.6%	12546.6	⁄
$n = 200$, `scaling`	100.0%	100.0%	1008.9	203.0	99.8%	99.8%	2078.2	208.0	100.0%	100.0%	5310.8	223.0	100.0%	100.0%	9824.1	*248.0*
$n = 500, \lambda_E = 0.05$	98.7%	98.0%	1149.8	518.1	84.9%	81.9%	1451.2	517.6	57.7%	56.1%	3602.3	*524.5*	70.1%	69.5%	22019.8	⁄
$n = 500, \lambda_E = 0.1$	99.3%	98.6%	873.9	538.1	90.5%	86.5%	908.7	*534.4*	60.5%	56.1%	1308.3	536.3	41.7%	40.1%	3725.3	*550.3*
$n = 500, \lambda_E = 0.2$	—	—	—	—	93.7%	88.8%	642.1	575.1	67.4%	58.2%	582.6	571.6	43.4%	37.1%	738.7	577.1
$n = 500, \lambda_V = 0.05$	98.7%	98.0%	1056.4	519.1	84.1%	81.4%	1244.6	518.1	55.5%	53.8%	3009.1	*524.8*	69.0%	68.5%	18871.9	⁄
$n = 500$, `scaling`	100.0%	100.0%	2505.7	503.0	100.0%	100.0%	5215.7	508.0	99.7%	99.7%	13234.9	*523.0*	100.0%	100.0%	26715.7	⁄
$n = 1000, \lambda_E = 0.05$	99.0%	98.3%	1935.5	*1035.9*	90.4%	87.5%	2241.7	*1032.8*	58.4%	55.0%	3006.2	*1034.9*	41.7%	40.6%	9908.8	⁄
$n = 1000, \lambda_E = 0.1$	99.6%	99.0%	1574.0	*1074.9*	93.8%	90.2%	1753.5	*1065.8*	67.5%	60.3%	1628.0	*1064.5*	43.6%	39.0%	2207.7	*1070.5*
$n = 1000, \lambda_E = 0.2$	—	—	—	—	94.9%	89.5%	1103.1	*1146.7*	74.7%	64.0%	1019.0	*1137.1*	50.1%	40.6%	924.4	*1137.2*
$n = 1000, \lambda_V = 0.05$	99.3%	98.7%	1808.1	*1037.4*	90.4%	87.3%	2200.1	*1033.9*	59.7%	55.9%	2793.2	*1036.0*	41.7%	40.5%	7904.4	*1050.1*
$n = 1000$, `scaling`	99.8%	99.7%	5105.9	*1003.0*	99.8%	99.8%	10082.5	⁄	99.8%	99.7%	25728.0	⁄	99.8%	99.8%	48757.2	⁄

Table 11.4: **Instance characteristics of `geom` graphs.** We use the same notation as in Table 11.2. Weights are given either as unit weights (denoted as 1^x) or exponential weights (2^x).

`geom`	$k = 5$				$k = 10$				$k = 25$				$k = 50$			
	$\frac{n'}{n}$	$\frac{m'}{m}$	χ	h	$\frac{n'}{n}$	$\frac{m'}{m}$	χ	h	$\frac{n'}{n}$	$\frac{m'}{m}$	χ	h	$\frac{n'}{n}$	$\frac{m'}{m}$	χ	h
$n = 1000, 1^x, \varrho = 1, \lambda_E = 0$	19.5%	10.3%	142.0	26.8	13.6%	6.2%	258.5	16.7	13.9%	4.5%	444.2	9.2	8.5%	1.8%	444.2	6.0
$n = 1000, 2^x, \varrho = 1, \lambda_E = 0$	10.1%	4.9%	55.8	33.2	7.3%	2.2%	66.8	22.1	5.4%	0.7%	34.2	13.6	4.4%	0.3%	46.5	8.1
$n = 1000, 1^x, \varrho = 5, \lambda_E = 0$	24.2%	12.0%	70.8	26.7	22.8%	10.2%	144.8	16.8	22.5%	8.7%	394.0	9.3	12.3%	2.5%	307.8	6.0
$n = 1000, 2^x, \varrho = 5, \lambda_E = 0$	16.1%	7.3%	33.2	29.2	13.9%	4.6%	37.0	22.0	10.3%	1.9%	34.0	14.4	8.0%	0.8%	38.0	7.7
$n = 1000, 1^x, \varrho = 0, \lambda_E = 0.2$	19.5%	11.1%	43.8	29.1	16.0%	7.0%	84.8	17.9	16.0%	4.9%	193.2	10.1	11.2%	2.3%	316.2	6.0
$n = 1000, 2^x, \varrho = 0, \lambda_E = 0.2$	13.8%	7.4%	30.2	33.0	9.8%	3.1%	29.0	22.0	9.3%	1.6%	49.2	15.4	5.5%	0.5%	23.2	8.0
$n = 1000, 1^x, \varrho = 1, \lambda_E = 0.2$	21.0%	11.9%	41.0	29.3	18.6%	8.4%	76.0	17.0	16.2%	5.3%	210.2	9.6	12.2%	2.4%	271.8	6.0
$n = 1000, 2^x, \varrho = 1, \lambda_E = 0.2$	15.4%	8.1%	28.2	31.7	12.4%	4.0%	28.2	22.3	8.3%	1.3%	30.8	14.2	5.7%	0.5%	33.5	7.9
$n = 1000, 1^x, \varrho = 5, \lambda_E = 0.2$	24.4%	11.6%	29.2	28.5	22.9%	9.8%	74.0	16.9	19.7%	6.3%	161.8	9.9	16.2%	4.2%	337.8	6.1
$n = 1000, 2^x, \varrho = 5, \lambda_E = 0.2$	20.8%	9.1%	22.2	32.0	15.7%	4.9%	18.2	21.5	12.4%	2.1%	18.0	14.8	6.7%	0.5%	15.8	7.6
$n = 1000, 1^x, \varrho = 0, \lambda_E = 0.5$	—	—	—	—	22.2%	10.0%	19.8	20.8	26.1%	8.1%	55.0	11.1	16.8%	4.5%	124.5	6.6
$n = 1000, 2^x, \varrho = 0, \lambda_E = 0.5$	—	—	—	—	15.8%	5.5%	11.2	24.7	10.9%	1.9%	9.8	13.8	8.0%	0.8%	10.5	8.6
$n = 1000, 1^x, \varrho = 1, \lambda_E = 0.5$	—	—	—	—	26.8%	10.6%	17.8	20.7	20.6%	5.6%	51.0	10.6	21.5%	4.6%	92.0	6.4
$n = 1000, 2^x, \varrho = 1, \lambda_E = 0.5$	—	—	—	—	16.2%	5.4%	9.8	23.8	10.6%	2.0%	12.8	13.5	8.5%	0.9%	11.5	8.4
$n = 1000, 1^x, \varrho = 5, \lambda_E = 0.5$	—	—	—	—	24.2%	9.3%	23.5	18.9	24.0%	6.9%	47.2	10.2	20.0%	4.6%	104.8	6.3
$n = 1000, 2^x, \varrho = 5, \lambda_E = 0.5$	—	—	—	—	19.0%	5.7%	10.2	23.8	11.9%	2.2%	9.5	13.6	8.9%	0.7%	8.8	8.0
$n = 1000, 1^x, \varrho = 0,$ `scaling`	23.8%	12.1%	105.2	30.0	29.4%	11.9%	142.8	19.6	26.8%	8.1%	233.8	10.8	20.2%	4.2%	444.5	6.5
$n = 1000, 2^x, \varrho = 0,$ `scaling`	19.0%	8.6%	48.8	34.7	20.1%	5.9%	46.2	23.0	20.4%	2.8%	30.5	16.1	9.2%	0.7%	44.5	9.2
$n = 1000, 1^x, \varrho = 1,$ `scaling`	28.3%	14.4%	83.8	28.5	34.9%	14.2%	102.2	18.7	32.5%	10.0%	356.8	10.3	25.6%	6.2%	703.8	6.7
$n = 1000, 2^x, \varrho = 1,$ `scaling`	28.9%	12.7%	48.8	35.8	23.2%	6.4%	42.2	25.6	20.2%	2.8%	26.2	16.5	12.3%	1.0%	47.2	8.8

$n = 1000$, 1^x, $\varrho = 5$, `scaling`	31.9%	14.2%	50.5	28.9	41.6%	15.6%	66.0	18.0	37.0%	12.2%	230.2	10.1	30.5%	6.2%	191.0	6.5
$n = 1000$, 2^x, $\varrho = 5$, `scaling`	28.1%	10.8%	24.5	34.8	24.9%	6.7%	25.8	23.5	23.7%	3.2%	20.0	15.7	15.0%	1.2%	23.2	9.3
$n = 2000$, 1^x, $\varrho = 1$, $\lambda_E = 0$	7.9%	4.0%	126.5	38.0	12.1%	4.9%	366.5	24.4	11.6%	3.9%	921.8	13.9	11.3%	2.9%	1411.0	8.9
$n = 2000$, 2^x, $\varrho = 1$, $\lambda_E = 0$	8.3%	4.1%	90.2	52.4	5.2%	1.5%	86.5	33.6	5.5%	0.8%	111.2	21.3	4.9%	0.4%	124.8	16.9
$n = 2000$, 1^x, $\varrho = 5$, $\lambda_E = 0$	17.0%	9.0%	117.0	39.4	17.4%	7.7%	251.0	24.9	22.8%	8.3%	983.8	14.7	10.0%	2.0%	559.5	9.0
$n = 2000$, 2^x, $\varrho = 5$, $\lambda_E = 0$	13.6%	6.4%	60.5	50.2	12.9%	4.2%	64.0	34.4	10.8%	2.1%	67.5	21.2	8.9%	1.0%	92.0	17.7
$n = 2000$, 1^x, $\varrho = 0$, $\lambda_E = 0.2$	11.8%	6.8%	45.0	39.8	18.6%	8.2%	232.5	26.4	12.3%	4.3%	270.8	14.5	14.1%	3.9%	968.2	9.5
$n = 2000$, 2^x, $\varrho = 0$, $\lambda_E = 0.2$	13.9%	7.5%	60.5	56.1	10.8%	3.4%	64.2	35.0	7.2%	1.1%	60.8	21.0	7.1%	0.6%	55.5	17.2
$n = 2000$, 1^x, $\varrho = 1$, $\lambda_E = 0.2$	16.2%	9.1%	53.8	41.3	16.4%	7.6%	126.2	26.2	19.7%	6.9%	365.5	14.9	17.6%	4.6%	337.8	9.5
$n = 2000$, 2^x, $\varrho = 1$, $\lambda_E = 0.2$	12.8%	7.1%	47.5	50.2	10.7%	3.6%	59.5	36.2	7.8%	1.3%	45.8	20.8	6.1%	0.6%	35.0	16.6
$n = 2000$, 1^x, $\varrho = 5$, $\lambda_E = 0.2$	20.2%	9.9%	42.0	40.9	19.9%	8.3%	89.2	26.4	21.0%	7.2%	371.5	14.9	17.0%	4.4%	750.5	9.0
$n = 2000$, 2^x, $\varrho = 5$, $\lambda_E = 0.2$	15.4%	7.1%	35.8	51.3	14.1%	4.8%	32.2	35.0	10.6%	1.8%	34.5	19.7	9.1%	0.9%	40.5	15.8
$n = 2000$, 1^x, $\varrho = 0$, $\lambda_E = 0.5$	—	—	—	—	19.6%	7.9%	26.5	28.7	21.3%	6.7%	73.5	15.9	22.6%	6.3%	204.0	9.9
$n = 2000$, 2^x, $\varrho = 0$, $\lambda_E = 0.5$	—	—	—	—	14.7%	4.8%	13.2	33.5	11.0%	1.7%	14.0	21.0	7.8%	0.7%	16.0	16.2
$n = 2000$, 1^x, $\varrho = 1$, $\lambda_E = 0.5$	—	—	—	—	20.4%	8.4%	31.2	27.8	18.6%	5.7%	72.0	15.0	21.8%	5.7%	245.5	10.2
$n = 2000$, 2^x, $\varrho = 1$, $\lambda_E = 0.5$	—	—	—	—	12.6%	4.4%	17.0	33.2	10.9%	1.8%	15.5	21.8	8.0%	0.7%	16.2	16.0
$n = 2000$, 1^x, $\varrho = 5$, $\lambda_E = 0.5$	—	—	—	—	20.0%	7.1%	28.0	27.1	20.6%	6.6%	85.8	15.0	17.8%	4.6%	210.5	9.6
$n = 2000$, 2^x, $\varrho = 5$, $\lambda_E = 0.5$	—	—	—	—	15.1%	4.7%	13.5	32.6	11.2%	1.8%	12.8	21.3	9.1%	0.9%	11.5	14.5
$n = 2000$, 1^x, $\varrho = 0$, `scaling`	19.4%	10.0%	129.2	43.3	27.7%	11.0%	188.8	28.9	33.0%	9.9%	351.8	16.2	31.3%	8.2%	753.0	10.2
$n = 2000$, 2^x, $\varrho = 0$, `scaling`	20.5%	9.2%	85.8	55.6	19.8%	5.5%	77.5	37.1	17.6%	2.3%	159.2	24.8	13.8%	1.0%	42.8	17.6
$n = 2000$, 1^x, $\varrho = 1$, `scaling`	19.4%	9.9%	154.8	43.5	24.6%	9.4%	208.8	27.7	29.2%	8.9%	596.2	16.4	26.9%	6.1%	1041.5	10.8
$n = 2000$, 2^x, $\varrho = 1$, `scaling`	18.8%	8.9%	80.8	57.1	23.8%	6.6%	76.5	36.8	20.4%	2.8%	71.2	23.3	15.7%	1.1%	70.5	18.1
$n = 2000$, 1^x, $\varrho = 5$, `scaling`	30.4%	13.0%	88.0	41.9	36.2%	12.8%	119.0	28.6	35.6%	9.7%	299.8	16.2	34.3%	7.6%	355.2	10.8
$n = 2000$, 2^x, $\varrho = 5$, `scaling`	28.5%	11.0%	46.5	53.9	26.6%	7.0%	40.2	36.0	19.7%	2.8%	33.8	22.2	16.6%	1.4%	33.2	17.3

$n = 5000$, 1^x, $\varrho = 1$, $\lambda_E = 0$	5.9%	3.0%	206.5	63.7	7.7%	3.3%	455.0	39.7	13.2%	4.5%	2153.2	25.1	9.5%	2.4%	2729.5	16.3
$n = 5000$, 2^x, $\varrho = 1$, $\lambda_E = 0$	5.9%	3.0%	214.0	80.2	7.2%	2.6%	393.2	66.1	6.0%	1.1%	401.8	53.8	4.3%	0.4%	283.5	24.3
$n = 5000$, 1^x, $\varrho = 5$, $\lambda_E = 0$	10.2%	5.3%	188.2	64.0	14.5%	6.2%	392.5	41.3	17.1%	6.2%	1263.8	24.2	14.6%	4.0%	2787.8	16.7
$n = 5000$, 2^x, $\varrho = 5$, $\lambda_E = 0$	9.6%	4.9%	160.0	75.8	12.3%	4.9%	263.0	67.1	8.6%	1.9%	294.0	46.3	8.5%	1.1%	284.8	22.7
$n = 5000$, 1^x, $\varrho = 0$, $\lambda_E = 0.2$	14.1%	7.8%	94.5	69.7	13.0%	5.6%	229.0	44.0	14.2%	4.9%	621.0	24.2	—	—	—	—
$n = 5000$, 2^x, $\varrho = 0$, $\lambda_E = 0.2$	10.8%	6.2%	98.8	80.5	8.0%	3.2%	138.5	62.0	7.2%	1.4%	193.2	45.9	—	—	—	—
$n = 5000$, 1^x, $\varrho = 1$, $\lambda_E = 0.2$	11.7%	6.3%	70.8	64.0	12.6%	5.5%	194.0	41.9	13.4%	4.6%	575.8	24.4	—	—	—	—
$n = 5000$, 2^x, $\varrho = 1$, $\lambda_E = 0.2$	11.7%	6.4%	82.0	81.6	11.5%	4.4%	160.0	62.5	9.6%	2.0%	190.0	45.2	—	—	—	—
$n = 5000$, 1^x, $\varrho = 5$, $\lambda_E = 0.2$	12.8%	6.5%	56.8	65.9	14.7%	6.2%	164.2	41.8	16.2%	5.9%	549.5	24.0	—	—	—	—
$n = 5000$, 2^x, $\varrho = 5$, $\lambda_E = 0.2$	12.9%	6.4%	69.2	79.1	13.1%	4.9%	121.2	63.4	11.5%	2.4%	133.0	46.2	—	—	—	—
$n = 5000$, 1^x, $\varrho = 0$, $\lambda_E = 0.5$	—	—	—	—	14.1%	5.7%	48.2	45.6	17.9%	5.2%	146.8	25.3	18.7%	5.0%	509.8	17.3
$n = 5000$, 2^x, $\varrho = 0$, $\lambda_E = 0.5$	—	—	—	—	12.5%	4.6%	30.8	60.4	10.1%	2.0%	43.8	40.1	8.2%	1.0%	36.8	21.3
$n = 5000$, 1^x, $\varrho = 1$, $\lambda_E = 0.5$	—	—	—	—	14.9%	5.8%	47.8	46.9	19.1%	5.5%	154.5	26.6	17.8%	4.2%	436.0	17.0
$n = 5000$, 2^x, $\varrho = 1$, $\lambda_E = 0.5$	—	—	—	—	12.4%	4.4%	25.8	62.2	11.2%	2.2%	38.2	40.5	9.0%	1.0%	45.2	22.1
$n = 5000$, 1^x, $\varrho = 5$, $\lambda_E = 0.5$	—	—	—	—	17.4%	6.1%	34.5	47.5	18.2%	5.2%	108.0	25.8	18.2%	3.9%	253.5	17.8
$n = 5000$, 2^x, $\varrho = 5$, $\lambda_E = 0.5$	—	—	—	—	12.9%	4.4%	29.2	61.8	12.5%	2.5%	47.5	40.5	9.2%	1.0%	34.8	21.6
$n = 5000$, 1^x, $\varrho = 0$, `scaling`	14.1%	7.0%	219.2	69.8	19.6%	7.5%	417.0	46.1	21.1%	5.9%	1878.2	26.1	25.8%	6.2%	2086.5	18.0
$n = 5000$, 2^x, $\varrho = 0$, `scaling`	16.4%	8.2%	152.5	83.1	20.5%	6.8%	224.5	69.5	17.8%	2.9%	274.8	48.4	14.0%	1.3%	207.8	24.5
$n = 5000$, 1^x, $\varrho = 1$, `scaling`	20.2%	10.3%	210.5	69.8	22.3%	9.2%	606.8	44.3	26.0%	7.5%	595.5	26.3	25.3%	7.3%	4169.2	17.9
$n = 5000$, 2^x, $\varrho = 1$, `scaling`	17.8%	8.8%	186.8	86.6	19.1%	6.3%	272.5	67.1	18.9%	3.2%	328.5	48.9	16.3%	1.5%	72.2	24.5
$n = 5000$, 1^x, $\varrho = 5$, `scaling`	20.9%	9.3%	137.2	67.3	25.5%	9.6%	273.5	46.3	25.9%	7.2%	869.2	26.5	32.4%	8.8%	1726.5	18.3
$n = 5000$, 2^x, $\varrho = 5$, `scaling`	23.6%	10.6%	115.8	84.9	24.2%	7.7%	171.5	67.5	24.9%	4.4%	249.2	50.8	18.3%	1.8%	144.2	24.2

12

ALGORITHMS

12.1 EXACT SOLUTIONS

To compute exact MSPath solutions, we propose a straight-forward integer linear programming (ILP) formulation. The preprocessing routine gives us the length L_i of any shortest s-t-path in G_i, for each $i \in [\tau]$. Furthermore, it lets us define $\delta_i^+(v) \subseteq E_i$ (or $\delta_i^-(v) \subseteq E_i$) as the number of edges that enter (leave, respectively) vertex v in a reduced instance. This allows us to use a directed flow formulation [Ord56] for assuring the path property.

For each stage $i \in [\tau]$, the binary variable $x_i(e)$ indicates whether edge $e \in E_i$ is in P_i. Constraints (12.1a) ensure that each P_i is an s-t-path, constraint (12.1b) forces P_i to be of shortest length. For each transition (G_i, G_{i+1}) the (de facto binary) variable $z_i(e)$ indicates—due to constraints (12.1c), (12.1d), and the objective function—whether edge $e \in E_i \cap E_{i+1}$ is in $P_i \cap P_{i+1}$. Thus, the objective is to maximize the transition quality.

$$\max \sum_{i \in [\tau-1]} \sum_{e \in E_i} z_i(e)$$

$$s.t. \sum_{e \in \delta^-(v)} x_i(e) - \sum_{e \in \delta^+(v)} x_i(e) = \mathbb{1}(v = s) - \mathbb{1}(v = t) \qquad \forall\, i \in [\tau],\ \forall\, v \in V \quad (12.1a)$$

$$\sum_{e \in E_i} w_i(e) \cdot x_i(e) = L_i \qquad \forall\, i \in [\tau] \quad (12.1b)$$

$$z_i(e) \leq x_i(e) \qquad \forall\, i \in [\tau - 1],\ \forall\, e \in E_i \cap E_{i+1} \quad (12.1c)$$

$$z_i(e) \leq x_{i+1}(e) \qquad \forall\, i \in [\tau - 1],\ \forall\, e \in E_i \cap E_{i+1} \quad (12.1d)$$

$$x_i(e) \in \{0, 1\} \qquad \forall\, i \in [\tau],\ \forall\, e \in E_i \quad (12.1e)$$

12.2 TWO-STAGE ALGORITHMS

In the following, we are going to make extensive use of the auxiliary algorithm `prefPath(`i, F`)`. It finds, among all shortest s-t-paths in G_i, a shortest s-t-path with the maximum number of edges from F. It does so by computing Dijkstra's algorithm w.r.t. the edge weights of E_i where the weight of the edges in $F \cap E_i$ is reduced by some small ε. See Theorem 33 for details, where it is discussed as the *preficiency algorithm* for MSPath.

We first present some algorithms for the 2-stage problem MSPath$|_2$, as these are later used as black-box algorithms for general MSPath.

Greedy (`G`)**:** Computes a shortest s-t-path $P_1 \leftarrow$ `prefPath(`$1, E_2$`)` in G_1 and, favoring this path, a shortest s-t-path $P_2 \leftarrow$ `prefPath(`$2, P_1$`)` in G_2.

Double Greedy (`Gd`)**:** Calls `G` twice independently: once as described above, then with the roles of the stages exchanged. The output is the solution with larger transition quality.

Iterated Greedy (`Gi`)**:** Computes (P_1, P_2) with `G` and then alternatingly reoptimizes $P_i \leftarrow \texttt{prefPath}(i, P_{3-i})$ for $i = 1, 2$ until the transition quality does not improve anymore.

Approximation (`A`)**:** This $1/\sqrt{2\chi}$-approximation algorithm (see Section 7.2) iteratively computes candidate solutions (pairs of paths) and finally outputs the pair with largest transition quality. Let $Y := E_1 \cap E_2$ be the initial set of edges to be preferred. In each iteration j, a pair of paths $P_1^{(j)} \leftarrow \texttt{prefPath}(1, Y)$ and $P_2^{(j)} \leftarrow \texttt{prefPath}(2, P_1^{(j)})$ is computed as a new candidate solution, and we update the set of preferred edges to $Y \leftarrow Y \setminus P_1^{(j)}$. According to the original description, the algorithm continues until eventually $Y = \varnothing$ (which is guaranteed to happen due to our preprocessing). Our implementation can furthermore correctly halt earlier if the current best transition quality matches the upper bound $|E_1 \cap \texttt{prefPath}(2, E_1)|$.

Double Approximation (`Ad`)**:** Similarly to how `Gd` doubles `G`, this variant calls `A` twice, the second time with the roles of the two stages exchanged. It outputs the solution with larger transition quality.

Bounded Approximation (`A5`)**:** A variation of `A` that halts after the first 5 candidate solutions (or earlier if `A` halts earlier).

12.3 MULTISTAGE ALGORITHMS

We consider two different polynomial-time approaches to find solutions for instances where $\tau > 2$.

Multistage Greedy (`M-G`)**:** After initializing $P_1 \leftarrow \texttt{prefPath}(1, E_2)$, subsequent paths $P_i \leftarrow \texttt{prefPath}(i, P_{i-1})$ are computed iteratively for $i = 2, \ldots, \tau$. Proceeding in the reversed direction, for each $i = \tau - 1, \ldots, 1$ the solution P_i is updated to $\texttt{prefPath}(i, P_{i+1})$. This process is repeated alternatingly front to back and back to front as long as the transition quality increases. Note that `M-G` coincides with `Gi` for $\tau = 2$.

Multistage with black-box (`B-`$*$)**:** This algorithm (see Section 7.2) uses any MSPATH$|_2$-algorithm $*$ as a black-box. The latter is executed on each consecutive pair of stages. Using a linear-time dynamic programming approach, it computes a collection of non-adjacent transitions whose transition qualities sum to the largest number. If the individual (2-stage) transitions are computed using some α-approximation, this routine yields an $\frac{\alpha}{2}$-approximation; in the case of `B-A` we thus obtain an $1/\sqrt{8\chi}$-approximation.

Improving over the description in Section 7.2 in practice, our implementation does not use arbitrary solutions for a stage that is neither optimized

to the previous nor to the next stage. Instead, considering such a stage G_i, we set P_i to either `prefPath(`i, P_{i-1}`)` or `prefPath(`i, P_{i+1}`)`, depending on which path yields the better transition qualities in conjunction with the solution paths of its neighboring stages (both of which are naturally fixed by the dynamic programming). We evaluate the algorithm's performance using each of the 2-stage algorithms described above as a black-box.

13 EXPERIMENTAL RESULTS

The different algorithmic variants differ mainly in their approach for solving two-stage sub-instances. Therefore it is natural to first investigate their performance on MSPath$|_2$ instances separately. Then, we consider the multistage instances with $\tau > 2$.

To better understand the performance of some algorithm `X`, we use the following notion: Given some instance, the *gap* is the ratio $(\texttt{opt} - \texttt{heu})/\texttt{opt}$, where `heu` is the objective value computed by `X` and `opt` the optimal objective value.

RESOURCES. All computations were run on an Intel Xeon Gold 6134 with 3.2 GHz and 256 GB RAM running Debian 9. We limit each run to a single thread with a 10-minute time limit. Our C++ (gcc 8.3.0) code uses OGDF Dogwood [CGJ+13] as a graph algorithms library; the code will become part of the next OGDF release. We use CPLEX 20.1 as our ILP solver.

13.1 MSPath$|_2$

To obtain MSPath$|_2$ instances, we simply use the first two stages of every instance of `grid`, `geom` and `hybr`, as well as the two-stage instances from `real` (which, by construction, are selected to have better non-triviality than a random stage pair in the 8-stage `real` instances). See Table 13.1 for some average key figures on the two-stage experiments.

Nearly all two-stage algorithms are able to find solutions for all MSPath$|_2$ instances within the time limit, except for `ILP`, which hits the time limit on 1.1% of the instances. In particular, due to the high success rate of `ILP` (whose few fails are restricted to very large `grid` instances), allows us to understand how often the heuristics and approximation algorithms yield optimal solutions as well. For `grid` and `geom` instances, the running times behave as one would expect on average: The greedy variants are fastest, followed by the `A` versions. The exact `ILP` is slower than the greedy approaches by up to 3 orders of magnitude (and still up to 2 orders of magnitude compared to `A` and `Ad`); only for `hybr` it is roughly 10-fold slower than the non-exact approaches. Naturally, `Gd` and `Ad` take about double the time of their basic counterparts. While `Gi` is slower than `G`, it is still faster than `Gd` on average: `G` and `Gd` require 2 and 4 calls to `prefPath`, respectively, but `Gi` typically terminates after the 3rd call, realizing that it cannot improve on the solution after the first two calls (i.e., the solution is identical to the one of `G`). Interestingly, `A5`'s running time is roughly comparable to that of `Gd`: it requires 2.64 iterations on average (and thus roughly 5 calls to `prefPath` on average, with a median of 2 calls), and does not suffer from outliers with a vast number of iterations as `A` does (see below). On the `hybr` benchmark set, `A`

Table 13.1: **Results for MSPath$|_2$ experiments:** The instances successfully solved by ILP yield a subset of each benchmark set for which we now know the optimal solutions. The "solved optimally" columns for ILP give the mean size of the respective subsets relative to the overall size of the benchmark sets. For the other algorithms, the values in the "solved optimally" columns, as well as the various "gap" columns, are then always given w.r.t. to these subsets. The columns "avg. gap" and "avg. gap (¬opt)" give the mean observed gaps to the optima, where the latter is restricted to the instances not solved to optimality by the considered algorithm. We suppress the "gap" columns for real, since all algorithms solved all these instances to optimality.

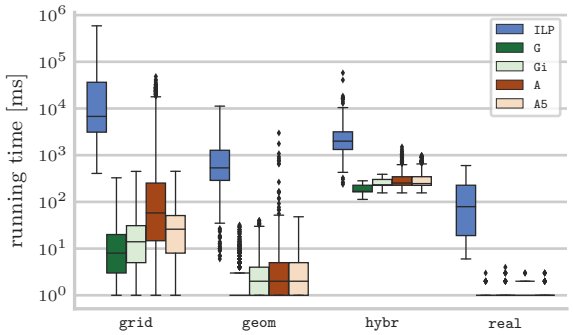
	time [ms]				solved optimally				avg. gap			avg. gap (¬opt)			max. gap		
	grid	geom	hybr	real	grid	geom	hybr	real	grid	geom	hybr	grid	geom	hybr	grid	geom	hybr
ILP	46787	897	2947	148	*98.9%*	*100%*	*100%*	*100%*	—	—	—	—	—	—	—	—	—
G	**20**	**3**	**185**	1	48.3%	95.0%	91.4%	100%	3.0%	0.8%	0.1%	5.9%	15.4%	0.9%	43.6%	37.5%	7.4%
Gd	41	5	316	1	56.2%	98.5%	**99.2**%	100%	1.5%	0.2%	0.0%	3.4%	11.2%	**0.3**%	23.7%	22.2%	**0.4**%
Gi	31	4	256	1	**70.4**%	96.9%	98.6%	100%	**1.0**%	0.5%	0.0%	**3.2**%	15.3%	0.7%	28.5%	33.3%	1.9%
A	972	17	361	1	48.9%	96.5%	91.4%	100%	2.6%	0.5%	0.1%	5.1%	13.0%	0.9%	27.4%	23.1%	6.0%
Ad	1913	34	660	2	56.6%	**99.0**%	**99.2**%	100%	1.4%	**0.1**%	0.0%	**3.2**%	**9.5**%	**0.3**%	**19.5**%	**16.7**%	**0.4**%
A5	46	4	346	1	48.9%	96.2%	91.4%	100%	2.6%	0.5%	0.1%	5.2%	13.1%	0.9%	27.4%	25.0%	6.0%

requires drastically fewer iterations than on grid and geom, and its running time becomes comparable to A5 and thus not too far off from the greedy approaches. The running times on the real instances are negligibly small for all algorithms, so we refrain from analyzing them in detail.

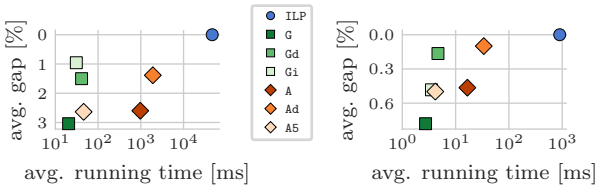
However, as depicted in Figure 13.1a, the average running times do not tell the whole story. While most algorithms expose a rather predictable running time, the high variance in the running time of A is stunning: for many grid and geom instances, A spends a lot of time on later iterations that only yield candidate solutions with trivially small transition quality, but is unable to deduce that further iterations are futile.

For the following quality comparisons of the non-exact algorithms, we only consider instances with known optimal objective value (i.e., those that ILP could solve to proven optimality). The 2-stage real instances can all be solved to optimality by all algorithms. We conclude that they are, despite our best effort, still too trivial. Also the hybr and geom instances can typically be solved to optimality by most algorithms, with success rates of (clearly) above 90%. In contrast to this, the grid instances yield comparably hard instances for the heuristics (seemingly independent of the precise parameter choices). Note that this is also the only set where ILP sometimes fails to prove optimal solutions (for $(x, y) \in \{50\} \times \{500, 1000\}$). Interestingly, Gi finds the maximum number of optimal solutions (79.7%) overall.

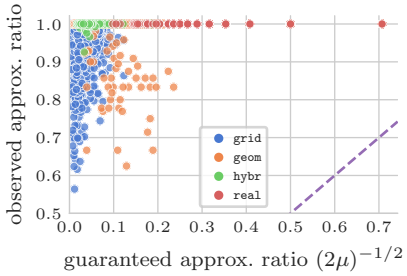
Considering the average gaps, however, the difference in hardness between grid and geom seems to flip: even though many grid instances are not solved optimally, the observed gaps are relatively low, within one-digit percentages. In



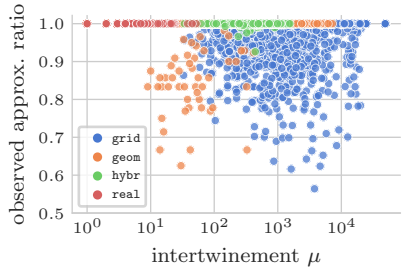
(a) Running times



(b) Trade-offs on `grid` (left) and `geom` (right)



(c) Approx. ratios: theory & practice



(d) Observed approx. ratio vs. χ

Figure 13.1: **Visualizations for the MSPath|2 experiments. (a)** The boxes show the median and quartiles; the whiskers extend to the farthest data point within 1.5 times the interquartile range. **(b)–(d)** We show the average gaps on all instances with known optimum; a gap g is equivalent to an observed approximation ratio of $1 - g$; the y-axes are arranged such that vertically higher data points represent solutions closer to the optimum.

contrast to this, non-optimally solved `geom` instances typically yield gaps in the range of 10%–15% for all non-exact algorithms.

The two greedy variants `Gd` and `Gi` beat `G` w.r.t. the objective value on 20.7% and 32.5% of the instances, respectively. The average improvement over the initial greedy objective value in these cases is 30.2% and 21.6%, respectively.

For `A`, the average number of iterations (each iteration requiring two calls to `prefPath`) is 35.3. However, the actual output solution is already found after 1.2 iterations on average, generating an average computational overhead of 60.7% per instance for futile subsequent iterations. In fact, for 95.6% of the instances `A` already finds the optimal solution with the initial candidate solution (which is the same solution `G` finds). For nearly all instances (99.2%), `A` finds its output solution within the first 5 iterations, i.e., here `A5` outputs the same solution as `A`. Inversely, even if `A5` terminates earlier than `A`, this yields worse solutions only in 2.1% of those instances. Using `Ad` improves the objective value compared to `A` on 19.9% of the instances; the average improvement is 31.2% (coming at the cost of doubling the running time).

Algorithm `A` has an approximation ratio of $1/\sqrt{2\chi}$. As Figure 13.1c shows, `A` not only performs much better than the worst-case analysis suggests, but the correlation between the observed approximation ratio (which is $1 - g$ for gap g) and the intertwinement χ is just not very pronounced on our instances (as shown more clearly in Figure 13.1d). Clearly, the quite intricate instance structures necessary to yield weak approximations do typical not appear in practice (at least not in our benchmark sets).

Figure 13.1b shows the trade-off between solution quality (in terms of average gap over the instances solved by `ILP`) and required running time for all considered algorithms. We can conclude that the `ILP` should be preferred if running time is not an issue, and one of the two greedy approaches `Gi` or `Gd` in all other cases. The slightly better running time of `G` is typically not worth it due to the drop in quality. One could also make a case for `Ad` which, despite requiring much more time, sometimes finds slightly better solutions than the greedy variants.

13.2 MSPath

Considering the true multistage instances, i.e., $\tau > 2$, we compare `M-G` and the various variants `B-{G,Gd,Gi,A,Ad,A5}`. Some key figures are presented in Table 13.2.

The `ILP`'s running times increase compared to the 2-stage scenarios, but not by as much as one might expect: compared to their 2-stage counterparts, the 16-stage `grid`, 16-stage `geom`, 4-stage `hybr`, and 8-stage `real` instances require roughly 3.5, 7.2, 1.4, and 4.5 times as much time, respectively. Thus, while `ILP` is certainly a time-wise expensive algorithm, we still can solve nearly all multistage instances to proven optimality within the time limit: it only fails on roughly 1/6 of the `grid` instances. This still allows us to investigate the ability of the non-exact algorithms to find optimal solutions.

Table 13.2: **Results for MSPath ($\tau > 2$) experiments:** Columns are interpreted as in Table 13.1. Recall that the grid, geom, hybr, and real instances have 16, 16, 4, and 8 stages, respectively.

	time [ms]				solved optimally				avg. gap (¬opt)				max. gap			
	grid	geom	hybr	real	grid	geom	hybr	real	grid	geom	hybr	real	grid	geom	hybr	real
ILP	165242	6453	4202	666	*84.4%*	*100%*	*100%*	*100%*	—	—	—	—	—	—	—	—
B-G	**206**	48	**627**	13	0.0%	1.6%	10.8%	35.0%	15.6%	13.8%	3.2%	10.4%	37.6%	45.2%	14.9%	20.8%
B-Gd	397	71	1023	19	0.0%	1.7%	11.4%	35.0%	14.7%	**13.4**%	3.2%	10.4%	37.6%	45.2%	**14.5**%	20.8%
B-Gi	338	58	839	16	0.0%	1.6%	11.1%	35.0%	**14.1**%	13.6%	3.2%	10.4%	37.6%	45.2%	**14.5**%	20.8%
B-A	13922	130	1185	13	0.0%	1.5%	10.8%	35.0%	15.5%	13.6%	3.2%	10.4%	37.6%	45.2%	14.9%	20.8%
B-Ad	19096	245	2107	22	0.0%	1.6%	11.4%	35.0%	14.6%	**13.4**%	3.2%	10.4%	37.6%	45.2%	**14.5**%	20.8%
B-A5	572	68	1121	16	0.0%	1.5%	10.8%	35.0%	15.5%	13.7%	3.2%	10.4%	37.6%	45.2%	14.9%	20.8%
M-G	257	**31**	692	**8**	**0.1**%	**2.1**%	**15.3**%	0.0%	16.8%	23.5%	3.2%	23.0%	45.1%	68.8%	23.7%	42.9%

First we may consider their running times. Observe that all B-$*$ variants first run their internal MSPath$|_2$ algorithm for $\tau - 1$ transitions. The subsequent dynamic programming over a sequence of only $\tau - 1$ integers requires negligible time compared to the various prefPath-calls. Hence, the running times of these algorithms are essentially the running times observed for their internal MSPath$|_2$ algorithms, scaled by the number of stage transitions. B-Ad is the only non-exact algorithm that (on 1.3% of the grid instances) runs into the time limit. The running time of M-G is very competitive and roughly comparable with the fastest B-$*$ variant, namely B-G.

The most interesting finding is how seldom the heuristics and the approximation approach find optimal solutions. While they all do so in most of the cases for MSPath$|_2$, the situation changes drastically for $\tau > 2$: We may start with discussing the B-$*$ variants, as they all yield essentially the same success rates: not a single multistage grid instance is solved to optimality (and only mediocre 19% and 11% of geom and hybr, respectively). Even for the previously too trivial real instances, the algorithms find optimal solutions only for roughly a third of the 8-stage instances. The reason for this consistent picture amongst all B-$*$ variants is easy to see: generally, the solution quality for the individual 2-stage sub-problems is very similar. Their common ingredient, i.e., the selection of "good" non-adjacent transitions, is to blame for the weak performance. While it is theoretically sound to simply essentially "ignore" every second transition (while still retaining an approximation ratio), this turns out to be abysmal in practice. In fact, we can see that this worst-case scenario is even (nearly) happening for some geom instances: despite the fact that half of the individual transitions are essentially optimal, we observe instances with an overall gap of 45.2%—very close to the worst case of 50%. Generally, this effect overshadows the influence of the precise selection of the black-box algorithm. Even when considering the gaps yielded by the non-optimal solutions, we only see slight deviations between the variants. Interestingly, the hybr instances allow generally lower gaps than the other benchmark sets.

Now, one may hope that the straight-forward but reasonably sounding heuristic `M-G` may fare better, but this is also hardly the case: it finds (only) two optimal solutions on `grid` instances and is slightly more successful than `B-*` on `geom` and `hybr`. For the `real` instances, however, it fails to find any optimal solution at all. Generally over all benchmark sets, its obtained gaps are weaker than those of `B-*`. In fact, on the `grid` instances its gaps can become close to 50% and for `geom` it even achieves a solution quality only 31.2% of the optimum (a gap of 68.8%).

Overall, we can see that no non-exact algorithm comes close to the optimal solution quality obtained by `ILP`, which is thus the probably best algorithmic choice—if time is not an issue. Otherwise, we would have to recommend the use of `B-G` or `M-G`, which are comparable in quality and running time. The other more expensive MSPATH$|_2$ algorithms are not worth it when used within the `B-*` context on these instances.

14

SUMMARY

In theory, the only known approximations for MSPath$|_2$ and general MSPath (A and B-A) guarantee approximation ratios of $1/\sqrt{2\chi}$ and $1/\sqrt{8\chi} = 1/(2\sqrt{2\chi})$, respectively, where B-A uses A internally and only differs by a constant factor of $1/2$. Thus, in the hunt for better (in particular constant) approximation ratios, it seems natural to focus on stronger approximations for MSPath$|_2$. However, our study shows that this is precisely *not* the interesting question when we want to obtain practically strong algorithms: MSPath$|_2$ is rather simple to solve in practice, the worst-case ratios of A are never met, and even simple greedy heuristics find close-to-optimal solutions. In contrast, the lifting from 2 to $\tau > 2$ stages is a central weak point which undermines the algorithms' success. Also, straight-forward alternative greedy strategies (M-G) do not work well. We therefore propose to focus on finding true multistage approximation routines, instead of relying on simple liftings from algorithms for few stages.

The general version of A from Part II is applicable to any preficient subgraph problem, which roughly speaking means that they allow a routine along the lines of prefPath. Thus, all G variants (as well as M-G) can also be used there, and we wonder if they perform similarly strong for other such problems as they do for MSPath$|_2$ (MSPath, respectively).

Part IV

PROSPECT

The stage is yours.
— Uri Geller: *The next Uri Geller* [Gelo8]

15

MAIN FINDINGS: A SUMMARY

In this thesis we investigated approximation algorithms for Multistage Subgraph Problems, some specialized for specific multistage problems and some generalized for a generic problem type. The focus was in particular to obtain good multistage solutions when the quality of per-stage solutions must not be compromised in favor of gains in the transition quality.

PART I. MULTISTAGE PERFECT MATCHINGS

In Part I, we studied two multistage variants of the classical matching problem, MIM and MUM. A core result is the $1/\sqrt{2\chi}$-approximation for MIM$|_2$ and its analysis. It is accompanied by several meta-results, which correlate (i) the two problems with one another and (ii) the 2-stage special case of the problems to their respective multistage formulation. They not only allow to make use of the positive algorithmic results, but also facilitate complexity considerations.

PART II. MULTISTAGE SUBGRAPH PROBLEMS

With little algorithmic effort, the algorithms for MIM$|_2$ and MIM could be generalized and fitted to the class of preficient Multistage Subgraph Problems (MSPs). The main contribution of Part II is the careful definition of the framework and the preficiency property, together with the general proof technique of well-behaved weight functions. The subsequent presentation of various preficient MSPs not only shows the simplicity of the framework's application, but also contains yet unpublished hardness results for such multistage problems.

PART III. EXPERIMENTAL STUDY: MULTISTAGE SHORTEST PATH

Part III is dedicated to an experimental study on the algorithmic behavior of the general algorithms presented in Part II. Using the illustrative example of MSPath, we assessed their performance regarding solution quality and running time compared to exact algorithms and heuristics. Aside from the experimental evaluation, another key contribution are the considerations on finding "hard" instances and discussions on suitable generators and parameter choices.

The following two chapters contain additional, yet unpublished results that clearly fall into the scope of this thesis but would have been misplaced in any of the previous parts. Together with the published results they provide the background for the concluding discussion of open questions.

16 FURTHER RESULTS ON MSPATH

The specific structure of MSPATH instances after the preprocessing of Section 10.3 allows to devise another approximation for MSPATH$|_2$ that is independent of the intertwinement (but still depends on an instance-specific parameter). This chapter is based on joint work with Fritz Bökler, Markus Chimani and Tilo Wiedera.

16.1 APPROXIMATION BY LONGEST PATH

The key for the MSPATH$|_2$ approximation Algorithm 50 is that any longest path in the intersection graph of a reduced MSPATH$|_2$ instance (which is a DAG, see Section 10.3) can be extended to a shortest s-t-path in each stage. Although finding a longest path is NP-hard in general graphs, on DAGs it can be solved in polynomial time [GJ79]. As we are mainly concerned with DAGs in this section, we will herein refer to directed edges as *arcs*.

Lemma 51. *In a reduced* MSPATH$|_2$ *instance, for any path Q in $G_\cap$ and any stage G_i there exists a shortest s-t-path P_i in G_i such that $Q \subseteq P_i$.*

Proof. Let $(a_1, ..., a_k)$ denote the arcs in Q in the order of a traversal. Since each stage G_i is reduced, there is a shortest s-t-path P_i in G_i that contains a_k. If $a_{k-1} =: uv$ is not in P_i, we may modify it by exchanging the subpath of P_i from s to v with a path from s to u plus a_{k-1}. The modified P_i is still a shortest s-t-path in G_i since all paths from s to v have the same length. We can now iterate this modification process for $a_{k-2}, ..., a_1$ to obtain a shortest s-t-path in G_i that completely contains Q. □

Since $G_\cap$ is in general not connected and an optimal MSPATH$|_2$ solution may be scattered all over $G_\cap$, we want to bound the error by which the optimum may deviate from a long path in $G_\cap$. To this end we will consider a decomposition of $G_\cap$ into components where the length of a longest path in such a component also provides a bound for the maximum number of edges that an optimal solution may realize in the respective component.

By the observations in Section 10.3, $G_\cap$ is a DAG that may possibly have multiple sinks and sources. We denote with σ^+ (σ^-) the number of *sources* (*sinks*,

Algorithm 50: Approximation for MSPATH$|_2$

1 compute a longest path Q in $G_\cap$
2 compute a shortest s-t-path P_i in G_i with $Q \subseteq P_i$ for $i = 1, 2$
3 **return** (P_1, P_2)

respectively) in $G_\cap$ after preprocessing, i.e., vertices with indegree (outdegree, respectively) zero. Further, let $\sigma := \min(\sigma^+, \sigma^-)$.

A *rooted arc set* (*RAS*) of a DAG $G = (V, A)$ is an arc set $B \subseteq A$ such that the subgraph $G[B]$ induced by B is a single-source DAG. The *depth* $d(B)$ of a RAS B is the length of a longest path in $G[B]$. A *rooted arc set decomposition* (*RASD*) of G is a partition $\mathcal{B} = \{B_1, ..., B_k\}$ of its arcs where each B_i is a RAS, i.e., a family of disjoint RASs such that $A = \bigcup_{i \in [k]} B_i$. Note that the number of sets in any RASD is at least σ^+, as no two sources can belong to the same RAS; a RASD with exactly σ^+ sets is *minimum*. The *depth* of a RASD $\mathcal{B}$ is $d(\mathcal{B}) := \max_{B \in \mathcal{B}} d(B)$; a RASD is *tall* if it maximizes the depth over all RASDs.

Lemma 52. *Let $G = (V, A)$ be a connected DAG. There is a RASD of G that is both tall and minimum.*

Proof. For each source v of G, create an initially empty RAS $B(v)$. Let P be a longest path in G and v_P its first vertex. By maximality of P, v_P is a source of G. Adding all arcs reachable from v_P to $B(v_P)$, we have $P \subseteq B(v_P)$ and thus $B(v_P)$ is a RAS of maximum depth. Similarly, for each source $v \neq v_P$, we greedily add to $B(v)$ all arcs that are reachable from v and are not in some RAS yet. As each arc is reachable from some source, the resulting RASD is minimum. □

The RASD concept allows us to reason about the length of a longest path.

Lemma 53. *Let opt denote the optimal solution value of a given* MSPATH$|_2$ *instance. For a longest path Q in the reduced $G_\cap$, we have $\mathsf{opt} \leq \sigma \cdot |Q|$.*

Proof. We give a detailed proof for $\mathsf{opt} \leq \sigma^+|Q|$; considering the instance that results from reversing all arcs, the rationale for sinks is analogous. Let $\mathcal{G} = (G_1, G_2)$ be the reduced multistage graph, (P_1^*, P_2^*) an optimal solution, and $P_\cap^* := P_1^* \cap P_2^*$ its intersection. Let $\mathcal{B}$ be a tall and minimum RASD of $G_\cap$ and let $B^* := \operatorname{argmax}_{B \in \mathcal{B}} |B \cap P_\cap^*|$ be a RAS in $\mathcal{B}$ with maximum intersection with $P_\cap^*$. By pigeonhole principle, we have $|B^* \cap P_\cap^*| \geq \mathsf{opt}/|\mathcal{B}| \geq \mathsf{opt}/\sigma^+$.

Let Q be a longest path in $G_\cap$. Since $\mathcal{B}$ is tall, we have $|Q| = d(\mathcal{B}) \geq d(B^*)$. The intersection $B^* \cap P_\cap^*$ consists of possibly disconnected subpaths of P_1^*, but its arcs all have distinct distances to s in G_1. Since arcs in B^* possess only $d(B^*)$ possible different distances to s in G_1, we have $d(B^*) \geq |B^* \cap P_\cap^*|$. Thus, $|Q| \geq |B^* \cap P_\cap^*| \geq \mathsf{opt}/\sigma^+$. □

Algorithm 50 builds a feasible solution where both paths contain the same longest path Q in the reduced $G_\cap$ as shown in Lemma 51. By Lemma 53, the intersection quality of this solution is thus at least opt/σ.

Theorem 54. *Algorithm 50 is a $1/\sigma$-approximation for* MSPATH$|_2$.

The above still holds for edge-weighted instances of MSPATH$|_2$. However, since the intersection quality only counts the number of common edges but not its weight, Q must still be computed as a longest path w.r.t. to unit costs.

For a MSPATH instance $\mathcal{G} = (G_i)_{i \in [\tau]}$ with $\tau > 2$, let σ_i denote the σ-value of the ith MSPATH$|_2$ subinstance (G_i, G_{i+1}) in $\mathcal{G}$. Let $\sigma_{\max} = \max_{i \in [\tau-1]} \sigma_i$ be the maximum over all consecutive MSPATH$|_2$ subinstances' σ-values.

Corollary 55. *Using Algorithm 50 with Theorem 25 is an* MSPATH *approximation with approximation guarantee* $2/\sigma_{\max}$.

16.2 REDUCTION TO MSPATH$|_2$

Instead of using Theorem 25, we can also reduce an MSPATH instance with an arbitrary number of stages to an equivalent MSPATH$|_2$ instance and then apply any appropriate (approximation) algorithm. The following result uses a technique similar to that used for MIM in Theorem 16.

Theorem 56. *There is an S-reduction from* MSPATH *to* MSPATH$|_2$*, i.e., given any* MSPATH *instance* $(\mathcal{G}, s, t)$*, we can construct a corresponding* MSPATH$|_2$ *instance* $(\mathcal{G}', s', t')$ *in polynomial time such that any solution for* $(\mathcal{G}, s, t)$ *bijectively corresponds to a solution for* $(\mathcal{G}', s', t')$ *with the same intersection quality and* $|E(G'_1) \cap E(G'_2)| = \sum_{i\in[\tau-1]} |E(G_i) \cap E(G_{i+1})|$.

Proof. We construct a 2-stage graph $\mathcal{G}'$ whose first stage G'_1 consists of (subdivided) concatenated copies of G_i for odd i; conversely its second stage G'_2 consists of (subdivided) concatenated copies of G_i for even i. More precisely, consider the following construction: Let $b(i) := 2 - (i \bmod 2)$. For each $i \in [\tau]$, we create a copy of G_i in $G'_{b(i)}$ where each edge $e \in E(G_i)$ is replaced by a 5-path p_i^e. We concatenate the copies in each stage by identifying the copy of t from G_i with the copy of s from G_{i+2} and set s' (t') to the copy of s (t) from the first (last) stage in G'_i.

We label the second (fourth) edge along p_i^e as e_i^- (e_i^+, respectively). To finally obtain $\mathcal{G}'$, for each $i \in [\tau - 1]$ and $e \in E(G_i) \cap E(G_{i+1})$, we identify the vertices of e_i^+ with those of e_{i+1}^- (disregarding the edges' orientations); thereby precisely the edges e_i^+ and e_{i+1}^- become an edge common to both stages. No other edges are shared between both stages. This completes the construction of $\mathcal{G}'$ and we have $|E(G'_1) \cap E(G'_2)| = \sum_{i\in[\tau-1]} |E(G_i) \cap E(G_{i+1})|$.

There is a natural bijection that links a solution $\mathcal{P}' = (P'_1, P'_2)$ for $(\mathcal{G}', s', t')$ to a solution $\mathcal{P} = (P_i)_{i\in[\tau]}$ for $(\mathcal{G}, s, t)$: the subpath of P'_1 through the subdivided copy of an odd stage G_i of $\mathcal{G}$ directly corresponds to a shortest s-t-path P_i in G_i.

Consider the intersection quality of $\mathcal{P}'$: Every edge in $P'_1 \cap P'_2$ corresponds to a different identification $\langle e_i^+, e_{i+1}^-\rangle$. We have $e \in P_i \cap P_{i+1}$ if and only if $e_i^- \in P'_{b(i)}$, $e_{i+1}^- \in P'_{b(i+1)}$, and $e_i^+ = e_{i+1}^-$. This in turn holds if and only if $e_i^+ \in P'_{b(i)} \cap P'_{b(i+1)}$ and hence the intersection qualities of $\mathcal{P}$ and $\mathcal{P}'$ are equal. □

The intertwinement $|E(G'_1) \cap E(G'_2)|$ of $\mathcal{G}'$ is largest w.r.t. the intertwinement of $\mathcal{G}$ if $|E(G_i) \cap E(G_{i+1})|$ is constant for all i. We thus obtain the following result.

Corollary 57. *Using Algorithm 28 after performing the reduction to a 2-stage instance yields an approximation ratio of* $1/\sqrt{2(\tau-1)\chi}$*; for* MSPATH$|_3$ *and* MSPATH$|_4$ *this is tighter than the* MSPATH *approximation via Algorithm 24.*

However, running Algorithm 50 on the new 2-stage instance would not yield any useful solution since its intersection graph consists solely of independent edges; the algorithm could thus only ensure a single common edge.

17

MULTISTAGE MINIMUM WEIGHT SPANNING TREE

All multistage problems considered in this thesis turned out to be NP-hard already on 2-stage graphs, even though their single-stage counterpart was polynomial-time solvable. However, we found one problem so simple that its 2-stage formulation is polynomial-time solvable with a basic algorithmic idea (similar to the one used in Algorithm 50). This chapter is based on joint work with Markus Chimani and Mirko H. Wagner.

In an undirected graph $G = (V, E)$ with edge weights $w\colon E \to \mathbb{N}$, a set of edges $T \subseteq E$ is a *spanning forest* of G if it is acyclic and of maximum cardinality. If T is connected, it is a *spanning tree*. A spanning tree T of G is *minimum weight* if the sum $\sum_{e\in T} w(e)$ of its edge weights is minimum over all spanning trees of G.

Definition 58 (MMST)**.** *Given an edge-weighted multistage graph $(G_i)_{i\in[\tau]}$, find a sequence $\mathcal{T} := (T_i)_{i\in[\tau]}$ of edge sets such that each T_i is a minimum weight spanning tree in G_i and the intersection quality $\sum_{i\in[\tau-1]} |T_i \cap T_{i+1}|$ is maximized. If there is an upper bound t on the number of stages τ,* MMST *is denoted by* MMST$|_t$.

Given uniform edge weights, any spanning tree is minimum weight. As such, one cannot make a wrong decision when iteratively building a spanning tree, which leads to the following result.

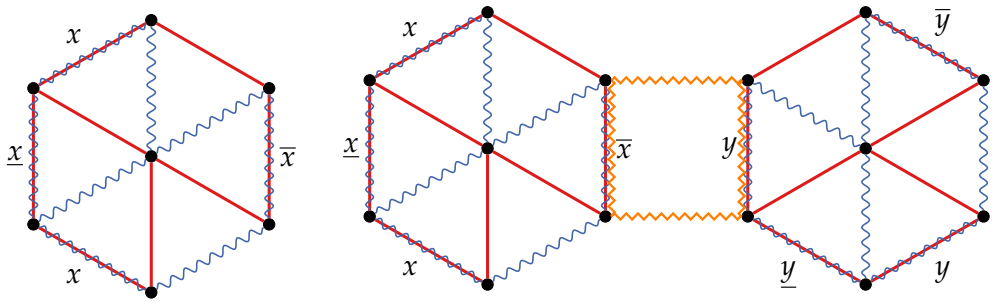
Theorem 59. MMST$|_2$ *with uniform edge-weights is solvable in polynomial time.*

Proof. Using Kruskal's algorithm [Kru56], we compute a *maximal* minimum weight spanning forest $T_\cap$ of $G_\cap$, i.e., an edge set that consists of a minimum weight spanning tree for each connected component of $G_\cap$. Starting from the intermediate solution $T_\cap$, Kruskal's algorithm allows to compute a minimum weight spanning tree T_i for each stage G_i such that $T_\cap \subseteq T_i$. Since $T_\cap$ has maximal cardinality, the intersection quality $|T_1 \cap T_2| = |T_\cap|$ is maximum. □

With a little bit of effort, this procedure can be generalized for an arbitrary number of stages that each have uniform edge weights; a rough proof can be found in [San23]. While the complexity of the weighted MMST$|_2$ problem also remains open, we can contrast above result with an NP-hardness result for the weighted problem on 3-stage graphs.

Theorem 60. MMST$|_3$ *is NP-hard, even when using only two weights $0 \leq w_1 < w_2$ consistently over all stages.*

Proof. In the NP-hard problem $(2,3)$-MaxSat [RRR98] we are given an integer $k \in \mathbb{N}$ and a set of boolean clauses on a variable set X, where each clause



(a) Variable gadget A_x (b) Clause gadget B_c for $c = \bar{x} \vee y$

Figure 17.1: Gadgets used in the proof of Theorem 60. E_1 is curvy blue, E_2 is straight red, and E_3 is zigzaggy orange; edges with a label have weight w_2, all other edges have weight w_1.

contains exactly 2 variables and each of the n variables in X occurs in at most 3 clauses. The question is whether there exists a variable assignment that satisfies at least k clauses. We may further assume that (after removing variables that only occur (non-)negated and possibly variable renaming) a variable $x \in X$ occurs at most twice non-negated and at most once negated.

Given an instance $\mathcal{I}$ of $(2,3)$-MaxSat, we construct an equivalent MMST$|_3$ instance $\mathcal{J}$ by generating a 3-stage graph (V, E_1, E_2, E_3) and setting the target value to $k' := 2n + k$. For each variable $x \in X$ we generate a gadget A_x as depicted in Figure 17.1a, where there are four edges with labels x, $\overline{x}$, and $\underline{x}$ that all have weight w_2 and eight further edges with weight w_1. These variable gadgets completely constitute G_1 and G_2. For clarity, we will now use $w_1 := 0$ and $w_2 := 1$ throughout the proof, but stress that any two weights $0 \leq w_1 < w_2$ evoke the same behavior.

In G_3, for each clause $c = \ell \vee \ell'$ with literals $\ell, \ell' \in \{x, \overline{x} \mid x \in X\}$ we generate a clause gadget B_c as depicted in Figure 17.1b: We choose an edge labeled ℓ and one labeled ℓ' from the respective variable gadgets, add them to E_3 and join them to a 4-cycle by adding two new weight-1 edges to E_3.

To ensure connectedness of each stage, add new vertices u_0, u_1, u_2, and u_3. For each $i \in [3]$, add an edge from u_i to each connected component in G_i. These new vertices are connected by edges u_0u_1, u_1u_2, u_2u_3 in G_1 and G_3, and by $\{u_0u_2, u_0u_3, u_1u_3\}$ in G_2.

Claim. *$\mathcal{J}$ is a yes-instance if and only if $\mathcal{I}$ is a yes-instance.*

Proof of Claim. We prove the two implications separately:

"$\Leftarrow$" Consider a yes-instance of $(2,3)$-MaxSat and a variable assignment that satisfies at least k clauses. We construct a 3-stage MST (T_1, T_2, T_3) for the corresponding MMST$|_3$ instance: In each stage we must necessarily choose all weight-1 edges to achieve minimality. Since none of these is present in two adjacent stages, they do not contribute to the intersection quality. In both T_1 and T_2, for each variable x we choose the two x-labeled edges if $x =$ `true` and the $\overline{x}$- and $\underline{x}$-labeled edges if $x =$ `false`. Thus, $T_1 \cap T_2$ contains exactly

$2n$ edges. In T_3, for each satisfied clause c choose an edge in $T_2 \cap B_c$; for each non-satisfied clause c choose an arbitrary labeled edge in B_c. As there are at least k satisfied edges, we have $|T_1 \cap T_2| + |T_2 \cap T_3| \geq 2n + k = k'$.

"$\Rightarrow$" Consider a $(2,3)$-MaxSat instance where the corresponding MMST$|_3$ instance allows a 3-stage MST $\mathcal{T} := (T_1, T_2, T_3)$ with intersection quality at least k'. Our goal is to show that the $(2,3)$-MaxSat instance is a yes-instance. As noted above, all weight-1 edges are necessarily in every MST for each stage. A variable gadget A_x is *consistent* in $\mathcal{T}$ if $T_1 \cap T_2$ contains either both x-labeled edges or both the $\overline{x}$- and the $\underline{x}$-labeled edge. By construction, A_x is consistent if and only if $|T_1 \cap T_2 \cap A_x| = 2$. Further, $T_i \cap A_x$ cannot contain three labeled edges for $i \in \{1,2\}$.

Suppose all variable gadgets are consistent; thus, we have $|T_1 \cap T_2| = 2n$ and $|T_2 \cap T_3| \geq k$. We derive a variable assignment by setting variable x to true if and only if the x-labeled edges are in $T_1 \cap T_2$. For each clause gadget B_c we can have at most one labeled edge in T_3 and $|T_2 \cap T_3 \cap B_c| = 1$ if and only if c is satisfied by the derived assignment. Thus, from $|T_2 \cap T_3| \geq k$ it follows that at least k clauses are satisfied by the derived assignment.

If any variable gadget is not consistent, we have $|T_1 \cap T_2| = 2n - d$ for some $d > 0$ and $|T_2 \cap T_3| \geq k + d$. Consider some inconsistent variable gadget A_x and adjust T_1 and T_2 such that it is consistent (it is irrelevant to the argument if x is consistently positive or negative). Since $|T_1 \cap T_2 \cap A_x| = 2$ only occurs for consistent variable gadgets, the adjustment increases $|T_1 \cap T_2|$ by at least 1. On the other hand, $|T_2 \cap T_3|$ can only decrease by at most 1, since at least one of the labeled edges in E_3 remains in T_2. After adjusting all d inconsistent variable gadgets, we have $|T_1 \cap T_2| = 2n$ and $|T_2 \cap T_3| \geq k$ and the above argument holds. ◁

This concludes the proof. □

18 FUTURE WORK

In this final section we point out possible directions for future research regarding multistage problems. While problem-specific open questions have already been posed in the summarizing chapter of the respective part, we add to those some more high-level questions.

APPROXIMATIONS. While we do not expect to find a polynomial-time exact algorithm for any of the NP-hard problems (unless P $\neq$ NP), our results could not completely rule out the existence of a polynomial-time approximation scheme or a constant-factor approximation.

1. Do any NP-hard MSPs allow for a constant-factor approximation?
2. Are there algorithms that exploit the characteristics of a particular MSP like Algorithm 50 does for MSPATH$|_2$?

As we discussed at the end of Chapter 6, our approximations are best suited for instances with low intertwinement, i.e., instances whose stages are mostly dissimilar.

3. Is there an approximation for any of the presented MSPs that is more powerful for instances with high intertwinement or otherwise highly similar stages?

MSPATH. The experimental study of Part III has been conducted with the goal to make educated guesses on the general behavior of MSPs. However, the MSPATH problem is interesting in its own right. An expanded experimental study that also considers the problem-specific algorithms of Chapter 16 could provide valuable new insights.

4. How does Algorithm 50 compare to the algorithms of Section 12.2?
5. How do the algorithms of Section 12.3 compare to an algorithm exploiting the S-reduction described in Theorem 56?

COMPLEXITY. In Chapter 17, we found that the complexity of MMST is not fully resolved yet. The respective open questions are particularly interesting as they could provide new insights in a general complexity dichotomy for multistage formulations of other "simple" polynomial-time solvable problems like, e.g., MAXIMAL MATCHING.

6. Is MMST with uniform edge weights NP-hard for more than two stages?

7. Is MMST$|_2$ still NP-hard with non-uniform edge weights?

8. Are there other multistage problems that are polynomial-time solvable when restricted to a fixed number of stages?

Our research was sparked by the problem of discerning the two optimization goals of the stage-wise objective and the transition quality. To further broaden our understanding of the interdependency between the two concurring objectives, one could consider the other natural edge case, where we do not fix the stage-wise objective, but the transition quality:

9. How do multistage problems behave where each transition is required to have optimal transition quality while per-stage solutions are allowed to be suboptimal?

Instead of fixing one objective to its optimum or investigating their weighted sum, one could also treat the optimization goals as two equally important objective functions in a multi-criteria optimization problem, where the goal is to compute Pareto-optimal solutions.

10. What insights can be found by considering an MSP as a two-dimensional multi-criteria optimization problem?

BIBLIOGRAPHY

[AGM+17] Eleni C. Akrida, Leszek Gąsieniec, George B. Mertzios, and Paul G. Spirakis. "The Complexity of Optimal Design of Temporally Connected Graphs." In: *Theory of Computing Systems* 61.3 (Oct. 1, 2017), pp. 907–944. DOI: 10.1007/s00224-017-9757-x.

[AMS+20] Eleni C. Akrida, George B. Mertzios, Paul G. Spirakis, and Viktor Zamaraev. "Temporal Vertex Cover with a Sliding Time Window." In: *Journal of Computer and System Sciences* 107 (Feb. 1, 2020), pp. 108–123. DOI: 10.1016/j.jcss.2019.08.002.

[AMS+21] Eleni C. Akrida, George B. Mertzios, Paul G. Spirakis, and Christoforos Raptopoulos. "The Temporal Explorer Who Returns to the Base." In: *Journal of Computer and System Sciences* 120 (Sept. 1, 2021), pp. 179–193. DOI: 10.1016/j.jcss.2021.04.001.

[BBR20] Julien Baste, Binh-Minh Bui-Xuan, and Antoine Roux. "Temporal Matching." In: *Theoretical Computer Science* 806 (Feb. 2, 2020), pp. 184–196. DOI: 10.1016/j.tcs.2019.03.026.

[BCE+23] Evripidis Bampis, Carmine-Emanuele Cella, Bruno Escoffier, Mila Rocco, and Alexandre Teiller. "Target-Based Computer-Assisted Orchestration: Complexity and Approximation Algorithms." In: *European Journal of Operational Research* 304.3 (Feb. 1, 2023), pp. 926–938. DOI: 10.1016/j.ejor.2022.05.008.

[Bea70] The Beatles. "The Long and Winding Road." In: *Let It Be* (1970). Lyrics by John Lennon and Paul McCartney. Published by Apple Records.

[BEK21] Evripidis Bampis, Bruno Escoffier, and Alexander Kononov. "LP-Based Algorithms for Multistage Minimization Problems." In: *Approximation and Online Algorithms*. Ed. by Christos Kaklamanis and Asaf Levin. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2021, pp. 1–15. DOI: 10.1007/978-3-030-80879-2_1.

[BEL+18] Evripidis Bampis, Bruno Escoffier, Michael Lampis, and Vangelis Th. Paschos. "Multistage Matchings." In: *16th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2018)*. Ed. by David Eppstein. Vol. 101. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2018, 7:1–7:13. DOI: 10.4230/LIPIcs.SWAT.2018.7.

[BES+21] Evripidis Bampis, Bruno Escoffier, Kevin Schewior, and Alexandre Teiller. "Online Multistage Subset Maximization Problems." In: *Algorithmica* 83.8 (Aug. 1, 2021), pp. 2374–2399. DOI: 10.1007/s00453-021-00834-7.

[BET22] Evripidis Bampis, Bruno Escoffier, and Alexandre Teiller. "Multistage Knapsack." In: *Journal of Computer and System Sciences* 126 (June 1, 2022), pp. 106–118. DOI: 10.1016/j.jcss.2022.01.002.

[BFK22] Robert Bredereck, Till Fluschnik, and Andrzej Kaczmarczyk. "When Votes Change and Committees Should (Not)." In: Thirty-First International Joint Conference on Artificial Intelligence. Vol. 1. July 16, 2022, pp. 144–150. DOI: 10.24963/ijcai.2022/21.

[BHI18] Sayan Bhattacharya, Monika Henzinger, and Giuseppe F. Italiano. "Deterministic Fully Dynamic Data Structures for Vertex Cover and Matching." In: *SIAM Journal on Computing* 47.3 (Jan. 2018), pp. 859–887. DOI: 10.1137/140998925.

[BKK+19] Robert Bredereck, Christian Komusiewicz, Stefan Kratsch, Hendrik Molter, Rolf Niedermeier, and Manuel Sorge. "Assessing the Computational Complexity of Multilayer Subgraph Detection." In: *Network Science* 7.2 (June 2019), pp. 215–241. DOI: 10.1017/nws.2019.13.

[BLS+14] Bartlomiej Bosek, Dariusz Leniowski, Piotr Sankowski, and Anna Zych. "Online Bipartite Matching in Offline Time." In: *2014 IEEE 55th Annual Symposium on Foundations of Computer Science*. 2014 IEEE 55th Annual Symposium on Foundations of Computer Science. Oct. 2014, pp. 384–393. DOI: 10.1109/FOCS.2014.48.

[BM23] Benjamin Merlin Bumpus and Kitty Meeks. "Edge Exploration of Temporal Graphs." In: *Algorithmica* 85.3 (Mar. 1, 2023), pp. 688–716. DOI: 10.1007/s00453-022-01018-7.

[BR04] Béla Bollobás and Oliver Riordan. "The Diameter of a Scale-Free RandomGraph." In: *Combinatorica* 24.1 (Jan. 1, 2004), pp. 5–34. DOI: 10.1007/s00493-004-0002-2.

[BS15] Aaron Bernstein and Cliff Stein. "Fully Dynamic Matching in Bipartite Graphs." In: *Automata, Languages, and Programming*. Ed. by Magnús M. Halldórsson, Kazuo Iwama, Naoki Kobayashi, and Bettina Speckmann. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2015, pp. 167–179. DOI: 10.1007/978-3-662-47672-7_14.

[Cas18] Arnaud Casteigts. "A Journey through Dynamic Networks (with Excursions)." thesis. Université de Bordeaux, June 4, 2018. URL: https://hal.science/tel-01883384 (visited on 05/12/2023).

[CCS22] Arnaud Casteigts, Timothée Corsini, and Writika Sarkar. "Invited Paper: Simple, Strict, Proper, Happy: A Study of Reachability in Temporal Graphs." In: *Stabilization, Safety, and Security of Distributed Systems: 24th International Symposium, SSS 2022, Clermont-Ferrand, France, November 15–17, 2022, Proceedings*. Berlin, Heidelberg: Springer-Verlag, Nov. 15, 2022, pp. 3–18. DOI: 10.1007/978-3-031-21017-4_1.

[CFQ+12] Arnaud Casteigts, Paola Flocchini, Walter Quattrociocchi, and Nicola Santoro. "Time-Varying Graphs and Dynamic Networks." In: *International Journal of Parallel, Emergent and Distributed Systems* 27.5 (Oct. 1, 2012), pp. 387–408. DOI: 10.1080/17445760.2012.668546.

[CGJ+13] Markus Chimani, Carsten Gutwenger, Michael Jünger, Gunnar W. Klau, Karsten Klein, and Petra Mutzel. "The Open Graph Drawing Framework (OGDF)." In: *Handbook on Graph Drawing and Visualization*. Ed. by Roberto Tamassia. Chapman and Hall/CRC, 2013, pp. 543–569.

[CL01] Fan Chung and Linyuan Lu. "The Diameter of Sparse Random Graphs." In: *Advances in Applied Mathematics* 26.4 (May 1, 2001), pp. 257–279. DOI: 10.1006/aama.2001.0720.

[CLX+21] Huanqing Cui, Ruixue Liu, Shaohua Xu, and Chuanai Zhou. "DMGA: A Distributed Shortest Path Algorithm for Multistage Graph." In: *Scientific Programming* 2021 (June 1, 2021), e6639008. DOI: 10.1155/2021/6639008.

[CPS21] Arnaud Casteigts, Joseph G. Peters, and Jason Schoeters. "Temporal Cliques Admit Sparse Spanners." In: *Journal of Computer and System Sciences* 121 (Nov. 1, 2021), pp. 1–17. DOI: 10.1016/j.jcss.2021.04.004.

[CRR+22] Arnaud Casteigts, Michael Raskin, Malte Renken, and Viktor Zamaraev. "Sharp Thresholds in Random Simple Temporal Graphs." In: *2021 IEEE 62nd Annual Symposium on Foundations of Computer Science (FOCS)*. 2021 IEEE 62nd Annual Symposium on Foundations of Computer Science (FOCS). Feb. 2022, pp. 319–326. DOI: 10.1109/FOCS52979.2021.00040.

[CT23] Markus Chimani and Niklas Troost. "Multistage Shortest Path: Instances and Practical Evaluation." In: *2nd Symposium on Algorithmic Foundations of Dynamic Networks (SAND 2023)*. Ed. by David Doty and Paul Spirakis. Vol. 257. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023, 14:1–14:19. DOI: 10.4230/LIPIcs.SAND.2023.14.

[CTW21] Markus Chimani, Niklas Troost, and Tilo Wiedera. "Approximating Multistage Matching Problems." In: *Combinatorial Algorithms*. Ed. by Paola Flocchini and Lucia Moura. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2021, pp. 558–570. DOI: 10.1007/978-3-030-79987-8_39.

[CTW22] Markus Chimani, Niklas Troost, and Tilo Wiedera. "Approximating Multistage Matching Problems." In: *Algorithmica* 84.8 (Aug. 1, 2022), pp. 2135–2153. DOI: 10.1007/s00453-022-00951-x.

[Dij59] Edsger W. Dijkstra. "A Note on Two Problems in Connexion with Graphs." In: *Numerische Mathematik* 1.1 (Dec. 1, 1959), pp. 269–271. DOI: 10.1007/BF01386390.

[Edm65a] Jack Edmonds. "Maximum Matching and a Polyhedron with 0,1-Vertices." In: *Journal of Research of the National Bureau of Standards Section B Mathematics and Mathematical Physics* 69B (1 and 2 Jan. 1965), p. 125. DOI: 10.6028/jres.069B.013.

[Edm65b] Jack Edmonds. "Paths, Trees, and Flowers." In: *Canadian Journal of Mathematics* 17 (1965), pp. 449–467. DOI: 10.4153/CJM-1965-045-4.

[EMS14] David Eisenstat, Claire Mathieu, and Nicolas Schabanel. "Facility Location in Evolving Metrics." In: *Automata, Languages, and Programming*. Ed. by Javier Esparza, Pierre Fraigniaud, Thore Husfeldt, and Elias Koutsoupias. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2014, pp. 459–470. DOI: 10.1007/978-3-662-43951-7_39.

[EP06] Nathan Eagle and Alex (Sandy) Pentland. "Reality Mining: Sensing Complex Social Systems." In: *Personal and Ubiquitous Computing* 10.4 (May 1, 2006), pp. 255–268. DOI: 10.1007/s00779-005-0046-3.

[Epp91] David Eppstein. "Offline Algorithms for Dynamic Minimum Spanning Tree Problems." In: *Algorithms and Data Structures*. Ed. by Frank Dehne, Jörg-Rüdiger Sack, and Nicola Santoro. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 1991, pp. 392–399. DOI: 10.1007/BFb0028278.

[EPY97] David Eppstein, Mike S. Paterson, and F. Frances Yao. "On Nearest-Neighbor Graphs." In: *Discrete & Computational Geometry* 17.3 (Apr. 1, 1997), pp. 263–282. DOI: 10.1007/PL00009293.

[EWF81] Earth, Wind and Fire. "The Changing Times." In: *Raise!* (1981). Lyrics by Bernard Taylor. Published by ARC/Columbia Records.

[FF56] Lester R. Ford Jr. and Delbert R. Fulkerson. "Maximal Flow Through a Network." In: *Canadian Journal of Mathematics* 8 (1956), pp. 399–404. DOI: 10.4153/CJM-1956-045-5.

[FG06] Jörg Flum and Martin Grohe. *Parameterized Complexity Theory*. Texts in Theoretical Computer Science. An EATCS Series. Berlin, Heidelberg: Springer, 2006. DOI: 10.1007/3-540-29953-X.

[FK22] Till Fluschnik and Pascal Kunz. "Bipartite Temporal Graphs and the Parameterized Complexity of Multistage 2-Coloring." In: *1st Symposium on Algorithmic Foundations of Dynamic Networks (SAND 2022)*. Ed. by James Aspnes and Othon Michail. Vol. 221. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022, 16:1–16:18. DOI: 10.4230/LIPIcs.SAND.2022.16.

[Flu21] Till Fluschnik. "A Multistage View on 2-Satisfiability." In: *Algorithms and Complexity*. Ed. by Tiziana Calamoneri and Federico Corò. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2021, pp. 231–244. DOI: 10.1007/978-3-030-75242-2_16.

[FMS09] Paola Flocchini, Bernard Mans, and Nicola Santoro. "Exploration of Periodically Varying Graphs." In: *Algorithms and Computation*. Ed. by Yingfei Dong, Ding-Zhu Du, and Oscar Ibarra. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2009, pp. 534–543. DOI: 10.1007/978-3-642-10631-6_55.

[FNR+22] Till Fluschnik, Rolf Niedermeier, Valentin Rohm, and Philipp Zschoche. "Multistage Vertex Cover." In: *Theory of Computing Systems* 66.2 (Apr. 1, 2022), pp. 454–483. DOI: 10.1007/s00224-022-10069-w.

[FNS+20] Till Fluschnik, Rolf Niedermeier, Carsten Schubert, and Philipp Zschoche. "Multistage S-t Path: Confronting Similarity with Dissimilarity in Temporal Graphs." In: *31st International Symposium on Algorithms and Computation (ISAAC 2020)*. Ed. by Yixin Cao, Siu-Wing Cheng, and Minming Li. Vol. 181. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2020, 43:1–43:16. DOI: 10.4230/LIPIcs.ISAAC.2020.43.

[Gel08] *The next Uri Geller – Unglaubliche Phänomene Live*. scriptwriter Mathias Taddigs. producer SevenOne Intermedia GmbH, Constantin Entertainment GmbH. ProSieben, 2008–2009.

[GJ79] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., 1979.

[GP81] Martin Grötschel and William R. Pulleyblank. "Weakly Bipartite Graphs and the Max-cut Problem." In: *Operations Research Letters* 1.1 (Oct. 1, 1981), pp. 23–27. DOI: 10.1016/0167-6377(81)90020-1.

[GTW14] Anupam Gupta, Kunal Talwar, and Udi Wieder. "Changing Bases: Multistage Optimization for Matroids and Matchings." In: *Automata, Languages, and Programming*. Ed. by Javier Esparza, Pierre Fraigniaud, Thore Husfeldt, and Elias Koutsoupias. Lecture Notes

in Computer Science. Berlin, Heidelberg: Springer, 2014, pp. 563–575. DOI: 10.1007/978-3-662-43948-7_47.

[Hås97] Johan Håstad. "Some Optimal Inapproximability Results." In: *Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing*. STOC '97. New York, NY, USA: Association for Computing Machinery, May 4, 1997, pp. 1–10. DOI: 10.1145/258533.258536.

[HHK+21] Klaus Heeger, Anne-Sophie Himmel, Frank Kammer, Rolf Niedermeier, Malte Renken, and Andrej Sajenko. "Multistage Graph Problems on a Global Budget." In: *Theoretical Computer Science* 868 (May 8, 2021), pp. 46–64. DOI: 10.1016/j.tcs.2021.04.002.

[HHS22] Kathrin Hanauer, Monika Henzinger, and Christian Schulz. "Recent Advances in Fully Dynamic Graph Algorithms." In: *1st Symposium on Algorithmic Foundations of Dynamic Networks (SAND 2022)*. Ed. by James Aspnes and Othon Michail. Vol. 221. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022, 1:1–1:47. DOI: 10.4230/LIPIcs.SAND.2022.1.

[HS19] Petter Holme and Jari Saramäki, eds. *Temporal Network Theory*. Computational Social Sciences. Cham: Springer International Publishing, 2019. DOI: 10.1007/978-3-030-23495-9.

[ISB+11] Lorenzo Isella, Juliette Stehlé, Alain Barrat, Ciro Cattuto, Jean-François Pinton, and Wouter Van den Broeck. "What's in a Crowd? Analysis of Face-to-Face Behavioral Networks." In: *Journal of Theoretical Biology* 271.1 (Feb. 21, 2011), pp. 166–180. DOI: 10.1016/j.jtbi.2010.11.033.

[KKK00] David Kempe, Jon Kleinberg, and Amit Kumar. "Connectivity and Inference Problems for Temporal Networks." In: *Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing*. STOC '00. New York, NY, USA: Association for Computing Machinery, May 1, 2000, pp. 504–513. DOI: 10.1145/335305.335364.

[Kru56] Joseph B. Kruskal. "On the Shortest Spanning Subtree of a Graph and the Traveling Salesman Problem." In: *Proceedings of the American Mathematical Society* 7.1 (1956), pp. 48–50. DOI: 10.1090/S0002-9939-1956-0078686-7.

[KRZ21] Leon Kellerhals, Malte Renken, and Philipp Zschoche. "Parameterized Algorithms for Diverse Multistage Problems." In: *29th Annual European Symposium on Algorithms (ESA 2021)*. Ed. by Petra Mutzel, Rasmus Pagh, and Grzegorz Herman. Vol. 204. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021, 55:1–55:17. DOI: 10.4230/LIPIcs.ESA.2021.55.

[Kun13] Jérôme Kunegis. "KONECT: The Koblenz Network Collection." In: *Proceedings of the 22nd International Conference on World Wide Web*. WWW '13 Companion. New York, NY, USA: Association for Computing Machinery, May 13, 2013, pp. 1343–1350. DOI: 10.1145/2487788.2488173.

[KV08] Bernhard Korte and Jens Vygen. "Combinatorial Optimization: Theory and Algorithms." In: *Springer, Third Edition, 2005*. (2008).

[KY04] Bryan Klimt and Yiming Yang. "The Enron Corpus: A New Dataset for Email Classification Research." In: *Machine Learning: ECML 2004*. Ed. by Jean-François Boulicaut, Floriana Esposito, Fosca Giannotti, and Dino Pedreschi. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2004, pp. 217–226. DOI: 10.1007/978-3-540-30115-8_22.

[LKF05] Jure Leskovec, Jon Kleinberg, and Christos Faloutsos. "Graphs over Time: Densification Laws, Shrinking Diameters and Possible Explanations." In: *Proceedings of the Eleventh ACM SIGKDD International Conference on Knowledge Discovery in Data Mining*. KDD '05. New York, NY, USA: Association for Computing Machinery, Aug. 21, 2005, pp. 177–187. DOI: 10.1145/1081870.1081893.

[LLD+08] Jure Leskovec, Kevin J. Lang, Anirban Dasgupta, and Michael W. Mahoney. *Community Structure in Large Networks: Natural Cluster Sizes and the Absence of Large Well-Defined Clusters*. Oct. 8, 2008. DOI: 10.48550/arXiv.0810.1355. arXiv: 0810.1355 [physics]. (Visited on 03/29/2023). preprint.

[LP86] László Lovász and Michael D. Plummer. *Matching Theory*. Elsevier, 1986.

[Mic16] Othon Michail. "An Introduction to Temporal Graphs: An Algorithmic Perspective*." In: *Internet Mathematics* 12.4 (July 3, 2016). DOI: 10.1080/15427951.2016.1177801.

[MMN+20] George B. Mertzios, Hendrik Molter, Rolf Niedermeier, Viktor Zamaraev, and Philipp Zschoche. "Computing Maximum Matchings in Temporal Graphs." In: *37th International Symposium on Theoretical Aspects of Computer Science (STACS 2020)*. Ed. by Christophe Paul and Markus Bläser. Vol. 154. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2020, 27:1–27:14. DOI: 10.4230/LIPIcs.STACS.2020.27.

[MMS19] George B. Mertzios, Othon Michail, and Paul G. Spirakis. "Temporal Network Optimization Subject to Connectivity Constraints." In: *Algorithmica* 81.4 (Apr. 1, 2019), pp. 1416–1449. DOI: 10.1007/s00453-018-0478-6.

[Mol20] Hendrik Molter. "Classic Graph Problems Made Temporal – a Parameterized Complexity Analysis." PhD thesis. Technical University of Berlin, Germany, 2020. URL: https://nbn-resolving.org/urn:nbn:de:101:1-202012090101228201737 4 (visited on 08/10/2023).

[Oet22] Lutz Oettershagen. "Temporal Graph Algorithms." Thesis. Universitäts- und Landesbibliothek Bonn, July 21, 2022. URL: https://bonndoc.ulb.uni-bonn.de/xmlui/handle/20.500.11811/10104 (visited on 08/15/2023).

[Ord56] Alex Orden. "The Transhipment Problem." In: *Management Science* 2.3 (Apr. 1956), pp. 276–285. DOI: 10.1287/mnsc.2.3.276.

[PBL17] Ashwin Paranjape, Austin R. Benson, and Jure Leskovec. "Motifs in Temporal Networks." In: *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining*. WSDM '17. New York, NY, USA: Association for Computing Machinery, Feb. 2, 2017, pp. 601–610. DOI: 10.1145/3018661.3018731.

[RRR98] Venkatesh Raman, B. Ravikumar, and S. Srinivasa Rao. "A Simplified NP-complete MAXSAT Problem." In: *Information Processing Letters* 65.1 (Jan. 15, 1998), pp. 1–6. DOI: 10.1016/S0020-0190(97)00223-8.

[RV89] Michael O. Rabin and Vijay V. Vazirani. "Maximum Matchings in General Graphs through Randomization." In: *Journal of Algorithms* 10.4 (Dec. 1, 1989), pp. 557–567. DOI: 10.1016/0196-6774(89)90005-9.

[San07] Piotr Sankowski. "Faster Dynamic Matchings and Vertex Connectivity." In: *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*. SODA '07. USA: Society for Industrial and Applied Mathematics, Jan. 7, 2007, pp. 118–126.

[San23] Jo Sandor. "Exploring the Complexity of Multistage Spanning Tree Problems." unpublished master's thesis, Osnabrück University, Germany, supervisors: M. Chimani and N. Troost. 2023.

[ST81] Daniel D. Sleator and Robert Endre Tarjan. "A Data Structure for Dynamic Trees." In: *Proceedings of the Thirteenth Annual ACM Symposium on Theory of Computing*. STOC '81. New York, NY, USA: Association for Computing Machinery, May 11, 1981, pp. 114–122. DOI: 10.1145/800076.802464.

[Tin23] *Tinder FAQ: Problems with Matches*. Match Group, LLC. URL: https://www.help.tinder.com/hc/en-us/articles/115003517666-Problems-with-Matches (visited on 08/10/2023).

[Tom60] Giuseppe Tomasi di Lampedusa. *The Leopard*. Time Reading Program Special Edition. New York: Time New York, 1960.

[Vaz03] Vijay V. Vazirani. *Approximation Algorithms*. Berlin, Heidelberg: Springer, 2003. DOI: 10.1007/978-3-662-04565-7.

[WCH+14] Huanhuan Wu, James Cheng, Silu Huang, Yiping Ke, Yi Lu, and Yanyan Xu. "Path Problems in Temporal Graphs." In: *Proceedings of the VLDB Endowment* 7.9 (May 2014), pp. 721–732. DOI: 10.14778/2732939.2732945.

[WS11] David P. Williamson and David B. Shmoys. *The Design of Approximation Algorithms*. Cambridge University Press, 2011. URL: http://www.cambridge.org/de/knowledge/isbn/item5759340/.

[XFJ03] 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* 14.02 (Apr. 2003), pp. 267–285. DOI: 10.1142/S0129054103001728.

[Yan78] Mihalis Yannakakis. "Node-and Edge-Deletion NP-complete Problems." In: *Proceedings of the Tenth Annual ACM Symposium on Theory of Computing*. STOC '78. New York, NY, USA: Association for Computing Machinery, May 1, 1978, pp. 253–264. DOI: 10.1145/800133.804355.