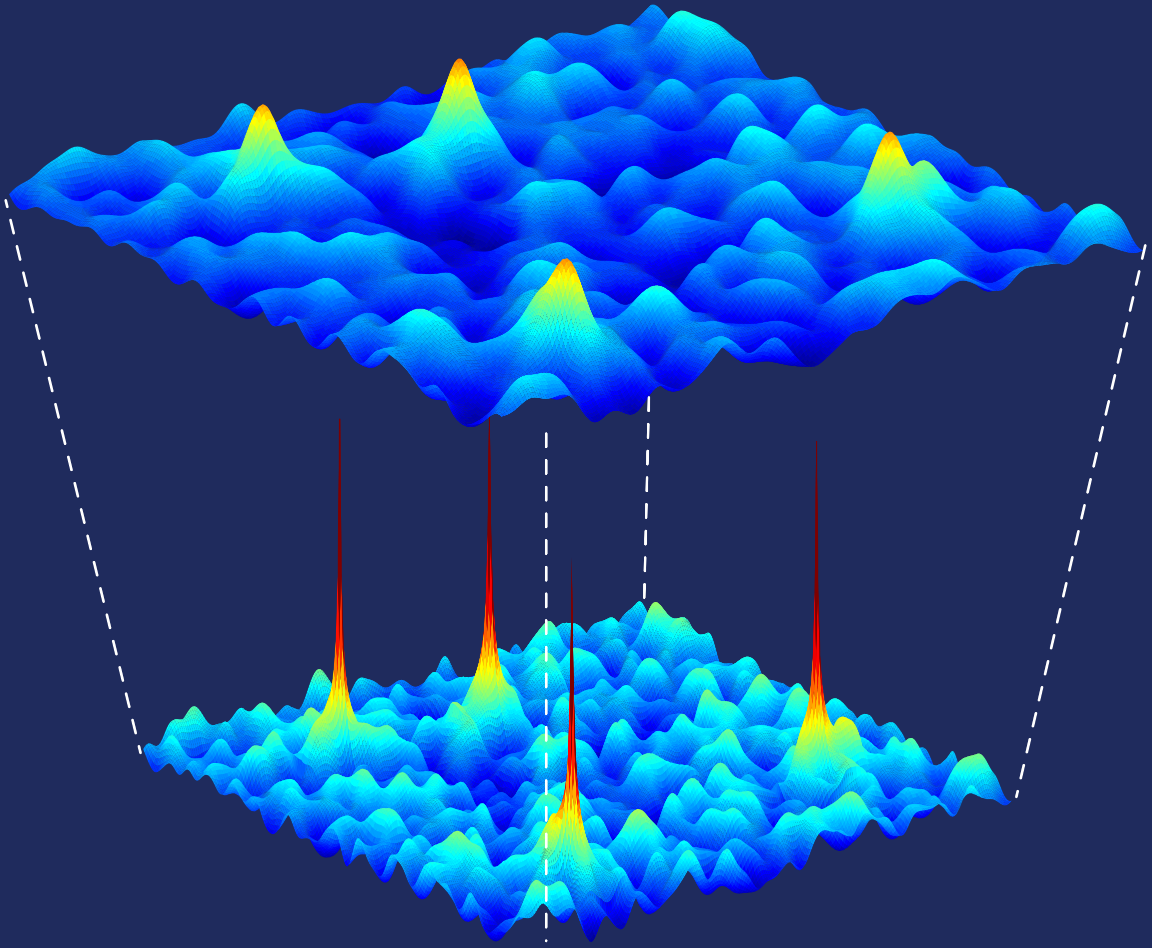


Smoothed analysis of belief propagation and minimum-cost flow algorithms

Kamiel Cornelissen

Smoothed analysis of belief propagation and minimum-cost flow algorithms

Kamiel Cornelissen

CTIT

CTIT Ph.D. Thesis Series No. 16-385
Centre for Telematics and Information Technology
University of Twente
P.O. Box 217, NL – 7500 AE Enschede

ISSN: 1381-3617 (CTIT Ph.D. Thesis Series No. 16-385)
ISBN 978-90-365-4097-1
DOI: 10.3990/1.9789036540971
http://dx.doi.org/10.3990/1.9789036540971

Typeset with LaTeX. Printed by Ipskamp Printing, Enschede, the Netherlands.

Cover design: Jikke Bakker.

This research was financially supported by The Netherlands Organisation for Scientific Research (NWO) grant 613.001.023.

SMOOTHED ANALYSIS OF BELIEF PROPAGATION AND MINIMUM-COST FLOW ALGORITHMS

PROEFSCHRIFT

ter verkrijging van
de graad van doctor aan de Universiteit Twente,
op gezag van de rector magnificus,
prof. dr. H. Brinksma,
volgens besluit van het College voor Promoties
in het openbaar te verdedigen
op vrijdag 27 mei 2016 om 16.45 uur

door

Kamiel Cornelissen

geboren op 14 december 1980
te Utrecht, Nederland

Dit proefschrift is goedgekeurd door:
Prof. dr. M.J. Uetz (promotor)
Dr. B. Manthey (copromotor)

Acknowledgments

At the start of a Ph.D. project, you never know where it will take you. Though the research plan for my project contained several paragraphs on the planned research for the first two years, the plan for the last two years consisted of only a couple of lines. Even this concise plan turned out to be too detailed, since some research directions turned out to be less promising than expected, while other interesting research opportunities appeared. This thesis is the result of all the research that I did during the last four years. I look back at my time as a Ph.D. student as a very enjoyable time, during which I learned many new things. Many people contributed to this, and I would like to thank them here.

First of all, I would like to thank my supervisors. Marc, thank you for giving me the opportunity to do research as a Ph.D. student in the DMMP group. Also, thank you for writing praising recommendation letters for me, which allowed me to participate in several interesting summer schools. Bodo, thank you for being my daily supervisor. It was a pleasure to be the first Ph.D. student under your supervision. Thank you for always having your door open for me and for coming to look for me when I would not visit you often enough. In addition, I am grateful for all the writing advice that you gave me to improve the quality of my papers.

While at the University of Twente, I had the pleasure to get to know many colleagues from the DMMP, SOR, and 'floor 3' groups of applied mathematics. Thank you all for always being willing to discuss research with me. Also, thank you for the fun times we had both at work and outside of work. Among others, I really enjoyed our pub quizzes, escape rooms, movie nights, kart racing afternoons, department outings, and games and beer nights at the Lunteren conferences. Thank you in particular to Ruben and Jasper, who were my office mates for most of my stay in the DMMP group. I always enjoyed our conversations, both concerning work topics and other topics (mostly games).

During my Ph.D. time I visited the University of Bonn several times. Heiko, Tobias, Clemens, and Michael, thank you for always making me feel welcome in Bonn and for the successful cooperation, from which two joint papers resulted.

Next to my scientific work at the University of Twente, I also had the opportunity to participate in several leisure activities. One of these was playing in the internal futsal league of the University of Twente. Thank you to all of my teammates of the three teams that I played for. I had a great time playing with you. First of all, the Tissue Regeneration team, which I joined when I performed the final project for my master's degree in the TR group and of which I was a member from (almost) the foundation of the team until the eventual disbanding of the team. Second, Pi

Hard. It was an honor being your number $(1+\sqrt{5})/2$. Finally, the Muppets. I have nice memories of all the times that we beat our opponents with our flawless passing game.

Besides playing futsal, I was a member of several more sports clubs. I had a great time and made many friends at these clubs. Thanks to all of you for making my time enjoyable, both on and off the field. Messed Up, it is nice to see how you have developed in the, by now, more than ten years after I had the pleasure to be involved in the foundation of the club. Fake Flamingo's, after so many years of trying, we finally got that championship. Thanks to Jikke in particular for taking care of the design of the cover of my thesis. Ludica, thank you for all the great matches, fun tournaments, and legendary Tuesday nights.

Over the years that I lived at Carpe Noctem I have seen many housemates come and go. One thing did not change, however. No matter the people that I lived with, we always had a good time and we spent many memorable evenings at the house bar. I fondly remember the Christmas dinners, Sinterklaas celebrations, house weekends, and all our other activities.

Many thanks go out to my family. Hans, Marjon, Jesse, and Stijn, thank you for always supporting me and for showing interest in my work, even though for most of you my research was far outside your expertise area. Jesse and Stijn, thank you for supporting me during the final hour of my time as a Ph.D. student by being my paranymphs. Stijn, thank you for the many comments on how to present my research more clearly. Your background in science communication was very helpful to me.

Finally, Irina, thank you for always being there for me. I hope that soon you will not be the only doctor in the house.

Contents

CHAPTER 1

Introduction

For many optimization problems, there is a large collection of algorithms that can be used to solve it. Suppose we have two algorithms A and B for a certain optimization problem P. Which of the two should we use to solve an instance of problem P? There can be many reasons to prefer one of the algorithms over the other. For example, algorithm A could be easier to implement, while algorithm B is more intuitive. One important aspect is the quality of the solution that an algorithm computes. Does the algorithm always compute the optimal solution to the optimization problem? If not, does it still provide a solution of reasonable quality?

Another important aspect is the time it takes an algorithm to compute a solution. Clearly, a fast algorithm is preferable over a slower algorithm. Usually, the running-time of an algorithm depends on the specific instance of the problem that is solved. It might be the case that algorithm A is faster for some instance I_1, while algorithm B is faster for some other instance I_2. Still, we would like to compare the speed of the two algorithms. A traditional way to do so is to perform a worst-case analysis of the running-time of the two algorithms.

In *worst-case analysis* of the running-time of an algorithm, we analyze the maximum time the algorithm requires for instances of a certain fixed size. A nice property of worst-case analysis is that it provides a guarantee for the maximum running-time of an algorithm. Given any instance, the size of the instance immediately gives an upper bound on the time the algorithm requires to solve it. A big disadvantage of worst-case analysis, however, is that it is often extremely pessimistic. A single instance for which the algorithm performs badly, can cause a very bad bound on the running-time. For most instances, the algorithm performs much better than the worst-case bounds suggest. Typical instances of a problem that are solved in practice often are very different from the artificial worst-case instances. For this reason, it would be nice to analyze the running-time of algorithms in a way that better matches the typical running-time of the algorithm, as opposed to the worst-case running-time.

Since we want to analyze the running-time of an algorithm for typical instances, the question rises what a typical instance of a problem looks like. This seems very problem-specific and hard to define in general. However, one property that most instances that are encountered in practice share, is that they are noisy. This noise can come from many sources. For example, the instruments used for measuring the data for the instance might be slightly inaccurate, or the computer used to process the data might have a limited numerical precision. Therefore, if it can be shown that an algorithm A works well for all instances that are subjected to a little noise, then A is also likely to work well for instances encountered in practice.

Smoothed analysis is a way of analyzing algorithms that is based on the above ideas. An imaginary adversary specifies any instance I of the problem. Subsequently, the instance I is slightly perturbed by adding a small amount of random noise to it before its running-time is analyzed. The *smoothed running-time* of an algorithm is the maximum expected running-time over all possible instances that the adversary can specify. Note that the adversary can specify some worst-case instance I_{WC} of the problem. However, the small amount of random noise that is added to I_{WC} is often enough to dramatically reduce the (expected) running-time of the algorithm. Many algorithms that have exponential worst-case running-time have polynomial smoothed running-time. This means that many worst-case instances are fragile and destroyed by adding a small amount of noise. Often, the smoothed running-time of an algorithm matches the time that the algorithm requires for instances encountered in practice much better than the worst-case running-time does.

In this thesis we rigorously analyze the performance of several algorithms. Most of the analysis is of one of two kinds. First, we analyze the quality of the solutions computed by the algorithms. We investigate whether an algorithm computes the optimal solution for all possible instances. If not, we show this by providing a counterexample. In addition, we investigate whether it does compute the optimal solution for certain classes of instances. Second, we analyze the smoothed running-time of the algorithms. We do so by proving lower and upper bounds for the smoothed running-time. The algorithms that we analyze are from two classes of algorithms: *belief propagation* (BP) algorithms and *minimum-cost flow* (MCF) algorithms.

The BP algorithm is a message-passing algorithm that can be used to solve probabilistic inference problems. Many problems can be modeled as probabilistic inference problems and BP has applications in lots of domains, such as machine learning, image processing, error-correcting codes, and statistics. The BP algorithm has recently enjoyed great popularity, since it is a simple and intuitive algorithm and often works well in practice. However, not much is known about the theoretical behavior of the BP algorithm. To get a better understanding of the BP algorithm, we apply it to well-known optimization problems such as the *maximum-weight matching* (MWM) problem, the MCF problem, the *maximum-weight independent set* (MWIS) problem, and the *minimum spanning tree* (MST) problem. Since the BP algorithm is an iterative algorithm, a natural question to ask is whether it always converges. If so, does it always converge to the optimal solution? What are lower and upper bounds for the running-time of the BP algorithm in the setting of smoothed analysis? In Chapters 2 and 3 we address these questions in a rigorous way.

The MCF problem is a well-known optimization problem, which has been studied for over half a century. The objective of the MCF problem is to send a certain amount of flow through a network in the cheapest possible way. Many problems can be modeled as an MCF problem, such as, for example, problems concerning transportation and communication networks. Over the last 50 years many algorithms have been developed that solve the MCF problem to optimality, and the running-time of these algorithms has been analyzed extensively. However, something interesting can be observed if we compare the worst-case running-time of these algorithms to the time that the algorithms require to solve instances from practice. The minimum-mean cycle canceling (MMCC) algorithm is strongly polynomial, while the successive shortest path (SSP) algorithm and the network simplex (NS) algorithm require an exponential number of iterations in the worst case. In sharp contrast with this, the SSP algorithm and the NS algorithm completely outperform the MMCC algorithm in experimental studies [37]. Since worst-case running-time bounds do not seem to give a good indication for the time required in practice by these MCF algorithms, we analyze them in the setting of smoothed analysis. In Chapters 4 and 5 we prove lower and upper bounds for the smoothed running-times of these algorithms.

In the rest of the introduction we introduce smoothed analysis (Section 1.1), the BP algorithm (Section 1.2), the MCF problem (Section 1.3), and the MWM, MWIS, and MST problems (Section 1.4) in more detail.

1.1 Smoothed Analysis

In this section we introduce smoothed analysis. Since in this thesis we mainly use the concept of smoothed analysis to analyze the smoothed running-time of algorithms, we focus on this aspect in the introduction.

Usually, the time required by an algorithm is measured in the size of the input instance. Larger instances typically require more computation time. Therefore, it makes sense to analyze the running-time of an algorithm for instances of some fixed size. The size of an instance can be, for example, the number n of nodes of the input graph or the number m of edges of the input graph. The dependence of the running-time of an algorithm on the size of the instance is often expressed using *big O notation*. Big O notation is used to specify how a function scales with the size of its arguments, omitting constant factors and lower-order terms. For an introduction to big O notation we refer to Cormen et al. [17].

In this thesis we make the common assumption that elementary operations such as adding, subtracting, multiplying, dividing, and comparing two numbers can be performed in one time step, even when the numbers are irrational. The main reason for having to make an assumption on the time required for performing elementary operations on irrational numbers is that, as we will see later, smoothed analysis uses a continuous perturbation model. This implies that with probability 1 the numbers in the perturbed instance are irrational.

In the above paragraph we mentioned that the running-time of an algorithm usually depends on the size of the instance, but we have left open how we define the running-time. It may be the case that the algorithm is much faster for some instance I_1 than for some other instance I_2, even though both have the same size. Therefore, it is not clear how we should link the running-time of an algorithm to the size of the instances. One choice one can make is to consider the worst possible instance for the algorithm among all instances of a certain fixed size n. This type of analysis

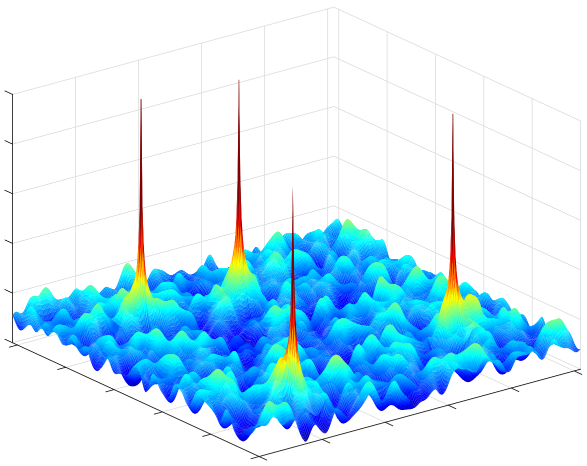


Figure 1.1: Typical dependence of the running-time of an algorithm on the input instances. Most worst-case instances are very fragile and their running-time is not a good indication for the running-time for most other instances.

is called *worst-case analysis*. In the following, let $\mathcal{I}$ be the set of all instances of a certain problem, and let $\mathcal{I}_n$ be the set of all instances of size n. We denote by $T_A(I)$ the running-time of algorithm A for instance $I \in \mathcal{I}$ (we usually omit the index A for simplicity of notation). The *worst-case running-time* $T_A^{\mathrm{WC}}(n)$ of algorithm A for instances of size n is defined as

$$T^{\mathrm{WC}}(n) = \max_{I \in \mathcal{I}_n} T(I).$$

Traditionally, worst-case analysis is the most popular way of analyzing the running-time of algorithms. An advantage of worst-case analysis is that it gives a guarantee for the maximum running-time of an algorithm. This guarantee is very strong if the worst-case running-time is low. However, for many algorithms the guarantee is extremely pessimistic. In Figure 1.1 we sketch a typical dependence of the running-time of an algorithm on the input instances. In the horizontal plane are all the instances of a certain size and along the vertical axis their running-time is plotted. Two things can be observed. First, for most instances the algorithm is much faster than the worst-case running-time suggests. Second, the worst-case instances are very fragile. A small perturbation of a worst-case instance suffices to dramatically decrease the running-time of the algorithm. For instances encountered in practice, there is usually no reason to assume that they are worst-case instances. Also, practical instances are often subject to a small amount of noise caused by, for example, measurement errors or rounding errors. Therefore, an algorithm is generally much faster for practical problems than it is in the worst case. Worst-case analysis is often not suitable to determine the usefulness of an algorithm in practice.

An alternative to worst-case analysis is *average-case analysis*. The *average-case* *running-time* $T_A^{\mathrm{AC}}(n)$ of algorithm A for instances of size n is defined as

$$T^{\mathrm{AC}}(n) = \mathop{\mathbb{E}}_{I \sim P_n}(T(I)),$$

where the instance I is drawn at random according to some fixed probability distribution P_n on the set of instances $\mathcal{I}_n$. In theory, the distribution P_n can be any distribution. Usually, simple distributions that are easy to analyze are used, such as the uniform distribution. Average-case analysis tries to capture how fast an algorithm is on average. An advantage of the average-case running-time over the worst-case running-time is that it is not completely determined by a small number of very bad instances. A big problem with average-case analysis, however, is that it is often not clear what distribution P_n to use to obtain typical instances of a problem that resemble instances encountered in practice. As an illustration, let us consider a problem that has a picture as its input. If we would construct an instance of this problem by assigning each pixel of the picture a color drawn uniformly at random, we obtain a picture similar to the image on the left in Figure 1.2. Almost everyone would agree that the result is not a very typical picture. The image on the right in Figure 1.2 is a much more typical example of a picture. Most pictures encountered in practice have large groups of neighboring pixels that are colored similarly, while the pictures generated using the uniform distribution usually lack such large groups. As another example, consider a problem that has as its input the locations of a number of cities on a map. If the locations of these cities are drawn uniformly at random, they are likely to be reasonably well-spread over the map. In contrast, if we consider, for example, the ten biggest cities in Canada, then we see that they are all located in the very south of the country.

The reason that average-case instances of a problem usually do not look like typical instances of the problem encountered in practice is that average-case instances often have special properties (like the lack of large groups of similarly-colored pixels in our picture example or the lack of large clusters of cities in our second example) with high probability. These special properties are often exploited in average-case analysis, where an algorithm is shown to work well for instances with these special properties. However, in many cases practical instances do not have these special properties. Therefore, the average-case running-time is not always a good indication for the running-time of an algorithm for practical instances.

Smoothed analysis was introduced by Spielman and Teng in 2001 to circumvent the problems of worst-case and average-case analysis [56]. They used smoothed analysis to explain the performance of the simplex method [22] for linear programming. (Their analysis was later improved and simplified by Vershynin [61].) Though the simplex method takes exponential time in the worst case [34], in practice it is about as fast as polynomial-time interior-point methods [60] and much faster than the polynomial-time ellipsoid method [9]. The reason for this is that worst-case linear programs are contrived and unlikely to occur in practice. In general, the worst-case running-time of an algorithm does not always provide a good indication for the speed of the algorithm for practical instances. The smoothed analysis framework is a way of analyzing algorithms that better matches performance in practice.

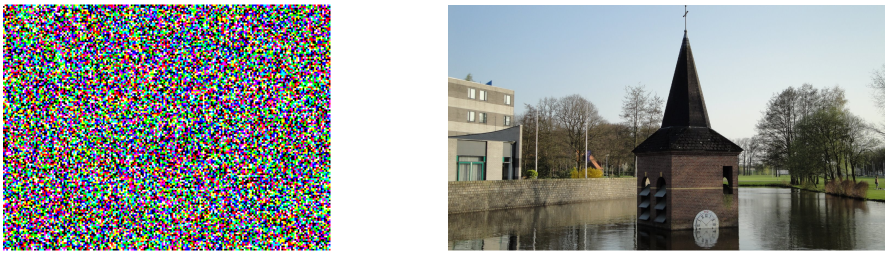


Figure 1.2: On the left an image where every pixel has a color drawn uniformly at random. On the right an image of ‘het Torentje van Drienerlo’ designed by Wim T. Schippers.

Smoothed analysis is a hybrid of worst-case and average-case analysis and an alternative to both. In the original model by Spielman and Teng, an adversary specifies an instance, and this instance is then slightly perturbed at random. In this way, pathological instances do not dominate the analysis. The assumption that an instance from practice is subject to some small perturbation is quite natural in many cases. The perturbation can model, for instance, measurement errors, numerical imprecision, or rounding errors. It can also model influences that cannot be quantified exactly, but for which there is no reason to believe that they are adversarial. We define the *smoothed running-time* $T_A^{\mathrm{Sm}}(n)$ of algorithm A for instances of size n as

$$T^{\mathrm{Sm}}(n) = \max_{J \in \mathcal{I}_n} \mathop{\mathbb{E}}_{I \sim P(J,\sigma)} (T(I)).$$

Here, $P(J, \sigma)$ is a probability distribution centered around the instance J with standard deviation σ, where σ is some small number. For example, $P(J, \sigma)$ could be the normal distribution with mean J and variance σ^2. Note that we have left some of the details of the above definition vague. These will be clarified in the rest of the thesis. A potential problem of the above definition is that the instance I obtained after perturbing J is not a valid instance of the problem anymore. To avoid this, in most cases where smoothed analysis is applied the structure of the instance is left intact. Only (a subset of) the numbers in the input is perturbed. For example, in case an instance consists of a weighted graph $G = (V, E)$, usually the node set V and the edge set E are left untouched and only the edge weights are perturbed.

The definition of the smoothed running-time mitigates some of the problems that we observed for worst-case and average-case analysis. The performance of an algorithm for a worst-case instance often dramatically improves when a little noise is added to the instance. Though the definition of the smoothed running-time includes taking the maximum over all instances J, including some worst-case instance J^{WC}, this instance J^{WC} is first perturbed before analyzing its (expected) running-time. Therefore, a few worst-case instances do not dominate the analysis of the smoothed running-time as much as they do for the worst-case running-time. A worst-case instance is not even necessarily bad in the smoothed setting. In fact, the instance J for which the expectation in the definition of the smoothed running-time is

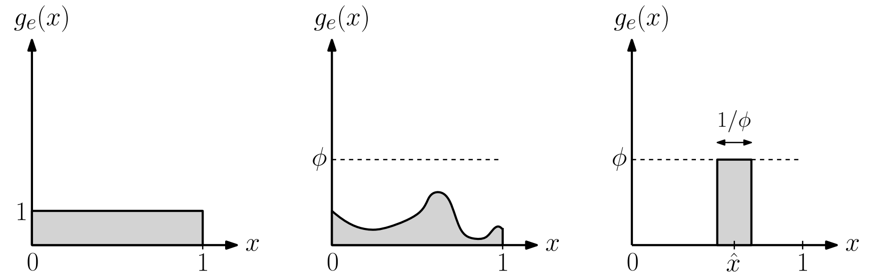


Figure 1.3: For the left image $\phi = 1$ and the adversary has no choice but to specify a uniform density. The middle image shows a valid choice for the density function when ϕ is larger than 1. Note that the density is bounded by ϕ for all $x \in [0, 1]$. The right image shows a density function that most resembles a worst-case choice. The value of x is drawn for sure from an interval of width $1/\phi$ around the value $\hat{x}$.

maximized is often not a worst-case instance. For average-case analysis we remarked that average-case instances often have some special property with high probability. If we perturb an instance J from practice that does not have this special property, it is likely to still not have this special property after perturbation, if the perturbation is sufficiently small. Therefore, the smoothed running-time is lower-bounded by the running-time for instances without the special property. This is in contrast to the average-case running-time. There these instances are barely influential, since they are vastly outnumbered by the instances with the special property. Good performance bounds of an algorithm in the smoothed setting usually indicate good performance for instances encountered in practice, since instances from practice are usually subject to a certain amount of noise. For this reason, smoothed analysis has been applied in a variety of contexts since its invention in 2001 [2, 6, 13, 21, 24, 47].

In this thesis, we follow a model of smoothed analysis due to Beier and Vöcking [7] that is slightly more general than the original model by Spielman and Teng [56]. In the model by Spielman and Teng, the adversary is allowed to pick any instance and each input parameter is subsequently perturbed according to a fixed distribution (the normal distribution). This model is often referred to as the two-step model of smoothed analysis. In the model by Beier and Vöcking, often referred to as the one-step model of smoothed analysis, the adversary is even allowed to specify the probability distribution of the perturbation. The power of the adversary is only limited by the *smoothing parameter* ϕ. The parameter ϕ determines the maximum density that the adversary can specify for the density functions that are used to draw the values of the input parameters. The larger ϕ, the more power the adversary has. In many settings it is natural to consider ϕ a constant. For example, consider an algorithm that requires as its input numbers that are measured by a device that typically makes measurement errors in the order of 1%. In this case, a value of $\phi = 100$ is reasonable.

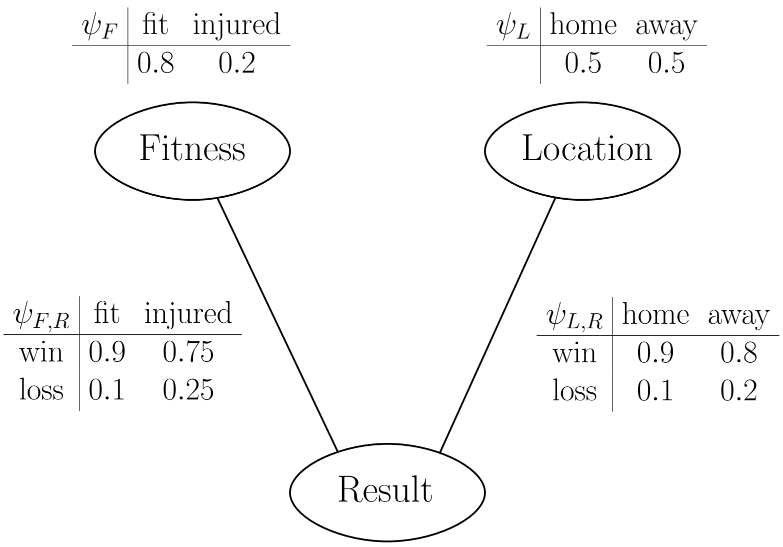
For concreteness, consider a problem where an instance consists of a graph $G = (V, E)$ with costs $c_e \in [0, 1]$ on the edges and the perturbation is only on the edge costs. In our input model the adversary does not fix the edge costs $c_e \in [0,1]$, but he or she specifies for each edge e a probability density function $g_e \colon [0,1] \to [0,\phi]$ according to which the costs c_e of the edge are randomly drawn independently of the other edge costs. Figure 1.3 shows three valid density functions for the edge costs. If $\phi = 1$, then the adversary has no choice but to specify a uniform distribution on the interval $[0,1]$ for each edge cost. In this case, our analysis becomes an average-case analysis. On the other hand, if ϕ becomes large, then the analysis approaches a worst-case analysis, since the adversary can specify a small interval I_e of width $1/\phi$ (which contains the worst-case costs) for each edge e from which the costs c_e are drawn uniformly at random. Another option for the adversary is to specify the density function of a truncated normal distribution with standard deviation $\sigma = O(1/\phi)$, resembling the original model by Spielman and Teng.

We refer to three recent surveys [38, 39, 57] for a broader picture of smoothed analysis.

1.2 Belief Propagation

In this section we introduce the *belief propagation* (BP) algorithm. Since the BP algorithm works on graphical models, we first introduce graphical models.

We start our introduction of graphical models with an example graphical model. Suppose we want to model the probability of FC Barcelona winning a football match (a match is decided by a penalty shoot-out if necessary, so draws are impossible). We assume that there are two main factors that influence the probability of FC Barcelona winning their match. First, is their star player Lionel Messi fit to play, or is he injured? Second, do they play a home match in Camp Nou, or do they play away? From previous experience we have some prior knowledge on these factors. FC Barcelona play half of their matches at home and half of their matches away. Also, Messi is injured 20% of the time. When Messi is playing, FC Barcelona win 90% of their matches, while they only win 75% when he is not playing. Finally, FC Barcelona win 90% of their home matches and 80% of their away matches. Using the prior knowledge, we can construct a joint probability distribution for this model. Let the random variable R with possible states $\{\text{win}, \text{loss}\}$ denote the result of the game for FC Barcelona. Also, let the random variable F with possible states $\{\text{fit}, \text{injured}\}$ denote the fitness of Lionel Messi. Finally, let the random variable L with possible states $\{\text{home}, \text{away}\}$ denote the location where the game is played. We encode the prior knowledge on the random variables in four functions ψ_F, ψ_L, $\psi_{F,R}$, and $\psi_{L,R}$. We call these functions *compatibility functions* (even the functions concerning a single variable). The functions ψ_F and ψ_L encode the a priori information about the fitness of Messi and the location of the match, respectively. The function $\psi_{F,R}$ encodes the compatibility of the various combinations of the fitness of Messi and the result of the match. The function $\psi_{L,R}$ encodes the compatibility of the various combinations of the location of the match and the result of the match. See Figure 1.4 for the values of the compatibility functions for all possible values of the corresponding random variables. The compatibility functions suggest the following factorization of the



ψ_F	fit	injured
	0.8	0.2

ψ_L	home	away
	0.5	0.5

$\psi_{F,R}$	fit	injured
win	0.9	0.75
loss	0.1	0.25

$\psi_{L,R}$	home	away
win	0.9	0.8
loss	0.1	0.2

Figure 1.4: The graphical model modeling the joint probability distribution of the result of a FC Barcelona match, the fitness of Messi, and the location of the match.

joint probability distribution P of the random variables F, L, and R

$$P(F = f, L = \ell, R = r) = \frac{1}{Z}\psi_F(f)\psi_L(\ell)\psi_{F,R}(f, r)\psi_{L,R}(\ell, r).$$

Here Z is a normalization constant, which is used to ensure that P is a valid probability distribution.

We can also model the dependence of the random variables using a graph. We call such a model a *graphical model*. In the graphical model, each node of the graph is associated with a random variable. Between two nodes there is an edge if and only if there is a compatibility function relating the corresponding variables. See Figure 1.4 for an illustration of the graphical model for the joint probability distribution P. Note that there is no edge between the variables corresponding to the fitness of Messi and the location of the match, since there is no prior knowledge that the two are dependent. For the graphical model there are many natural questions to ask, such as: What is the probability that FC Barcelona win their match? What is the probability that FC Barcelona are playing an away match, given that they lose their match? What is the most likely combination of values that the random variables F, L, and R can take? These kind of problems are called *probabilistic inference* problems.

Typically, a lot of computation time is required to solve these probabilistic inference problems. For example, even computing the value of the normalization constant Z requires summing an exponential number of terms, and it is not clear how to do this more efficiently. Many probabilistic inference problems are known to be NP-hard. For this reason, heuristics are often used to approximately solve probabilistic inference problems. The belief propagation algorithm is such a heuristic. It was proposed by Pearl in 1988 [45]. Belief propagation is a message-passing algorithm that is used for solving probabilistic inference problems on graphical models. Recently, BP has experienced great popularity. It has been applied in many fields, such as machine learning, image processing, computer vision, and statistics. There are two main reasons for the popularity of BP. First, it is widely applicable and easy to implement because of its simple and iterative message-passing nature. Second, it performs well in practice in numerous applications [58, 64].

Before we describe the BP algorithm in more detail, we first provide the graphical model that we consider in the rest of this introduction. Suppose we are given a graph $G = (V, E)$ with $V = \{1, 2, \ldots, n\}$ and for each $u \in V$ an associated random variable X_u that takes values in a finite set $\mathcal{X}_u$. We define $\mathcal{X} = \mathcal{X}_1 \times \mathcal{X}_2 \times \ldots \times \mathcal{X}_n$. Consider the probability distribution

$$\hat{P}(\mathbf{x}) = \frac{1}{Z} \left(\prod_{u \in V} \psi_u(x_u) \right) \left(\prod_{(u,v) \in E} \psi_{uv}(x_u, x_v) \right), \quad \mathbf{x} = (x_v)_{v \in V} \in \mathcal{X}. \tag{1.1}$$

In the above, the ψ_u and ψ_{uv} are non-negative functions and Z is a normalization constant. The graph G and the probability distribution $\hat{P}$ together form a graphical model, in particular a pairwise Markov random field (MRF). Since most of the problems considered in this thesis can be modeled as problems on pairwise MRFs, we restrict ourselves to pairwise MRFs in this introduction. This is not a big restriction, since other graphical models such as Bayesian networks and factor graphs can be converted to pairwise MRFs, though sometimes at the cost of introducing new random variables with possibly large state spaces [65]. Therefore, for BP algorithms defined on Bayesian networks and factor graphs, we can define equivalent BP algorithms on pairwise MRFs.

There are several variants of the belief propagation algorithm. The two best-known variants are the sum-product variant and the max-product (sometimes also called min-sum) variant of belief propagation. The sum-product variant is used to compute marginal distributions. In this thesis we consider the max-product variant of BP and at all places where we refer to the BP algorithm we mean the max-product variant of BP. The max-product variant of BP is used to compute maximum *a posteriori* probability (MAP) estimates. A *maximum a posteriori probability* (MAP) estimate of a probability distribution P is a most likely realization of the random variables. That is, the MAP estimate $\mathbf{x}^\star$ of $P(X)$ is defined as

$$\mathbf{x}^\star \in \operatorname{argmax} P(\mathbf{x}).$$

In the following we assume that the MAP estimate is unique. We call the value $x_u^\star$ that x_u takes in the MAP estimate the *MAP assignment* of u.

Computing the MAP estimate is NP-hard [55] for general probability distributions. The BP algorithm is a heuristic for computing the MAP estimate on graphical models. It is a message-passing algorithm on the graph G representing the graphical model. For the probability distribution $\hat{P}$ (see Equation (1.1)), BP computes the MAP estimate exactly when the graph G is a tree. In this case, the BP algorithm is equivalent to dynamic programming and the algorithm terminates after a number of iterations equal to the diameter of the tree. However, if G contains cycles, BP is not guaranteed to compute the correct MAP estimate. The reason for this is that messages sent by a node u can travel along a cycle to end up back at node u. This causes u to receive back the information in the message that it sent itself, wrongly increasing its conviction that this information is correct. But, even in the presence of cycles in the graph G, the BP algorithm is still well-defined and in practice often gives a good approximation of the MAP estimate.

In short, the BP algorithm works as follows. In each iteration k, each node u sends a message vector

$$M^k_{u\to v} = \left(m^k_{u\to v}(x_v)\right)_{x_v\in\mathcal{X}_v}$$

to each node v in its neighborhood $N(u) = \{v \mid \{u,v\} \in E\}$ containing a message for each possible value for X_v. A message $m^k_{u\to v}(x_v)$ can be interpreted as how "likely" the sending node u thinks it is that the random variable X_v associated with the receiving node v should take value x_v in the MAP estimate. The greater the value of the message $m^k_{u\to v}(x_v)$, the more likely it is according to node u in iteration k that X_v should take value x_v in the MAP estimate. The messages are initialized neutrally, that is, in iteration 0 the messages are

$$M^0_{u\to v} = (1, 1, \ldots, 1), \text{ for all } u, v \in N(u).$$

In iterations $k \geq 1$ the messages are computed from the messages in the previous iteration as follows:

$$m^k_{u\to v}(x_v) = \max_{x_u\in\mathcal{X}_u}\left\{\psi_u(x_u)\cdot\psi_{uv}(x_u,x_v)\cdot\prod_{w\in N(u)\setminus\{v\}} m^{k-1}_{w\to u}(x_u)\right\}.$$

This means that the sending node u determines the likelihood that X_v should take value x_v by computing its own value x_u that is most compatible with x_v, taking into account its local information ψ_u, the function $\psi_{u,v}$ describing the compatibility of values x_u and x_v, and the messages received from its neighbors other than v. The message received from v is not used, since the information contained in this message is already known by v and sending it back to v would cause the information to be doubly counted.

The belief b^k_u of node u in iteration k is defined as

$$b^k_u(x_u) = \psi_u(x_u)\cdot\prod_{v\in N(u)} m^{k-1}_{v\to u}(x_u).$$

These beliefs can be interpreted as the "likelihood" that X_u should assume value x_u in the MAP estimate. The greater the value of $b^k_u(x_u)$, the more likely that X_u should take value x_u in the MAP estimate. Node u computes its belief using its local information ψ_u and the messages received from its neighbors. Though the beliefs indicate the likelihood that a random variable assumes a certain value in the MAP estimate, they do not have a natural interpretation as a probability distribution. We denote the best estimate (breaking ties arbitrarily) for the value of X_u in the MAP estimate during iteration k by x_u^k, that is,

$$x_u^k = \operatorname{argmax}\{b_u^k(x_u) \mid x_u \in X_u\}.$$

The vector $(x_u^k)_{u \in V}$ gives an estimate of the MAP estimate during iteration k. If, for some k_0, we have

$$(x_u^{k_1})_{u \in V} = (x_u^{k_0})_{u \in V}, \quad \text{for all } k_1 \geq k_0,$$

then BP has converged after k_0 iterations. In general there are three possibilities: BP converges to the MAP estimate, BP converges to an incorrect solution, or BP does not converge at all. In particular, if the MAP estimate is not unique, then BP usually does not converge. The reason for this is that a node u for which X_u does not take the same value in all MAP estimates, does not know what to "believe". Typically, the estimate x_u^k for this node switches between multiple values in this case. Therefore, we assume in this thesis that the MAP estimate is unique.

We conclude our introduction of BP by considering BP applied to several combinatorial optimization problems (Section 1.2.1) and by introducing computation trees (Section 1.2.2), which are a very useful tool to analyze the BP algorithm. For a more elaborate introduction to BP and several of its applications, we refer to Yedidia et al. [65] and Mooij [41].

1.2.1 BP Applied to Combinatorial Optimization Problems

As we remarked before, if the graphical model is tree-structured, BP computes exact MAP estimates. However, if the graphical model contains cycles, then the convergence and correctness of BP have been shown only for specific classes of graphical models. To improve the general understanding of BP and to gain new insights about the algorithm, it has recently been tried to rigorously analyze the performance of BP as either a heuristic or an exact algorithm for several combinatorial optimization problems. Amongst others, it has been applied to the maximum-weight matching (MWM) problem [3, 5, 50, 51], the minimum spanning tree (MST) problem [4], the minimum-cost flow (MCF) problem [28], the maximum-weight independent set (MWIS) problem [52,53], and the 3-coloring problem [16]. In addition, Even and Halabi [25] have applied BP to packing and covering problems, extending and unifying some results obtained for the MWM and MWIS problems. The reason to consider BP applied to these combinatorial optimization problems is that these optimization problems are well understood. This facilitates a rigorous analysis of BP, which is often difficult for other applications.

Many optimization problems can naturally be modeled as graphical models. Consider, for example, the maximum-weight independent set problem (see also Sections 1.4.2 and 3.1). For each node u, we define a binary variable X_u. We encode the objective function of the MWIS problem in the single-variable compatibility functions, by setting $\psi_u = e^{w_u x_u}$. In addition, we encode the constraint that two neighboring nodes cannot both be in an independent set in the two-variable compatibility functions $\psi_{u,v}$ by setting $\psi_{u,v}(1,1) = 0$ and $\psi_{u,v}(x_u, x_v) = 1$ for all $(x_u, x_v) \neq (1,1)$. Now, the original input graph and the probability distribution $\hat{P}$ (see Equation 1.1) together form a graphical model for which the MAP estimate corresponds to the optimal solution of the original MWIS problem.

Similarly, many optimization problems, including linear programming, can be modeled as graphical models. Sometimes though, we have to allow continuous state spaces for the random variables. We refer to Gamarnik et al. [28] for more details.

In the remainder of this section we summarize some results that have been obtained for BP applied to combinatorial optimization problems. In addition, we state our results which are contained in this thesis. For more details on the BP algorithms for each of the problems and on the results we refer to Chapters 2 and 3.

Bayati et al. [5] have shown that the BP algorithm correctly computes the maximum-weight matching (MWM) in bipartite graphs if the MWM is unique. Belief propagation can also be used for finding maximum-weight perfect matchings in arbitrary graphs and finding maximum-weight perfect b-matchings [3, 51]. In this case, though, the BP algorithm only converges if the relaxation of the corresponding linear program has an optimal solution that is unique and integral. In all cases, the convergence of the BP algorithm takes pseudo-polynomial time and depends linearly on both the weight of the heaviest edge and $1/\delta$, where δ is the difference in weight between the best and second-best matching. In Chapter 2 we analyze the BP algorithm for MWM in the setting of smoothed analysis. We show that the probability that the BP algorithm needs more than k iterations is upper bounded by $O(nm\phi/k)$. In addition, we show that there exist instances for which the probability that the BP algorithm needs more than k iterations is lower bounded by $\Omega(n\phi/k)$.

Gamarnik et al. [28] have shown that BP can be used to find a minimum-cost flow, provided that the instance has a unique optimal solution. The number of iterations until convergence is pseudo-polynomial and depends linearly on the reciprocal of the difference in cost between the best and second-best integer flow. In Chapter 2 we analyze the BP algorithm for MCF in the smoothed setting and show an upper bound of $O(n^2 m\phi/k)$ for the probability that the BP algorithm for MCF needs more than k iterations. We also show that there exist instances for which the probability that the BP algorithm needs more than k iterations is lower bounded by $\Omega(n\phi/k)$.

Sanghavi et al. [53] have shown that the BP algorithm applied to the maximum-weight independent set problem does not converge if the LP relaxation of the instance has a non-integral optimal solution. Also, they have shown that even if the LP relaxation of the problem has a unique integral optimal solution, the BP algorithm is not guaranteed to converge. In Chapter 3 we extend this result by characterizing precisely the graph structures for which the BP algorithm is guaranteed to converge to the correct solution irrespective of the node weights (as long as the MWIS is unique). We show that the graphs for which the BP algorithm converges to the correct solution for all possible node weights are exactly those graphs that contain at most one even cycle and no odd cycles.

Bayati et al. [4] have shown that if the BP algorithm applied to the minimum spanning tree problem converges, then it converges to the correct solution. However, in Chapter 3 we show a small instance for which the BP algorithm does not converge. In addition, the property of this instance that ensures that the BP algorithm does not converge is quite general and carries over to many other instances. Therefore, we believe that the BP algorithm does not converge for most instances of the MST problem in practice.

1.2.2 Computation Tree

To show several of our results in Chapters 2 and 3, we need the notion of a *computation tree*. Computation trees have been used frequently to analyze the BP algorithm, for example, in the context of the maximum-weight independent set problem [53] and the maximum weight matching problem [3].

Let $G = (V, E)$ be an arbitrary undirected graph. We denote the *level-k computation tree* with the root labeled $u \in V$ by $T^k(u)$. In the following we call the root of a computation tree the CT-root, to distinguish it from the root of a directed spanning tree, which we introduce in Chapter 3. The tree $T^k(u)$ is a labeled rooted tree of height $k+1$. Like Bayati et al. [4] we denote by $[x, u]$ a node x in the computation tree with label u. In the rest of this thesis we will use the term u-labeled to denote that a node in the computation tree is labeled with node $u \in V$ and the term S-labeled to denote that a node in the computation tree is labeled with a node from the subset $S \subset V$. Also, we will refer to an edge between a u-labeled node and a v-labeled node in the computation tree as a $\{u, v\}$-labeled edge.

The CT-root in $T^0(u)$ has label u, its degree is the degree of u in G, and its children are labeled with the adjacent nodes of u in G. The tree $T^{k+1}(u)$ is obtained recursively from $T^k(u)$ by attaching nodes to every leaf node in $T^k(u)$. To each leaf node $[y, v]$ in $T^k(u)$, a number of nodes equal to the degree of v in G minus 1 is attached. These nodes are labeled with the neighbors of v in G except for the label of the parent of y in $T^k(u)$. If the nodes or edges of G are weighted, these weights are copied to the computation tree. This means that a u-labeled node in the computation tree has weight $w(u)$ and a $\{u, v\}$-labeled edge in the computation tree has weight $w(u, v)$. Figure 1.5 shows an example of an edge-weighted graph and computation tree.

The definition of the computation tree is such that each non-leaf node $[x, u]$ in the computation tree has neighbors with the same labels as the neighbors of u in G. Also, the messages that the u-labeled CT-root of a level-k computation tree receives after k iterations of the BP algorithm on the computation tree are exactly the same as the messages that u receives after k iterations of the BP algorithm on G. The behavior of the BP algorithm on trees is well understood, in contrast to the behavior of the BP algorithm on graphs with cycles. Therefore, computation trees form a useful tool for analysis of the BP algorithm on graphs with cycles.

On a computation tree $T = (V_T, E_T)$ we can naturally define a probability dis-

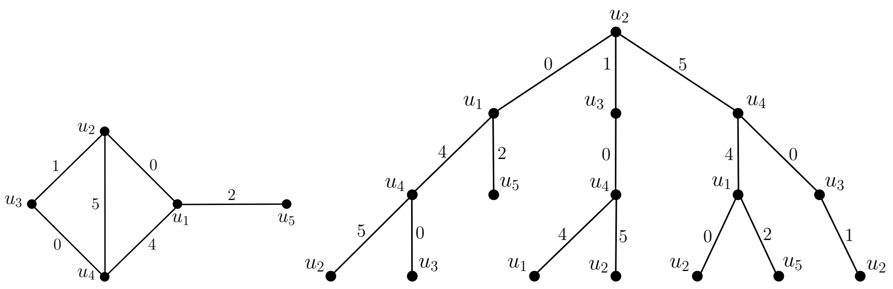


Figure 1.5: On the left an example edge-weighted graph and on the right the associated level-2 computation tree $T^2(u_2)$ rooted at u_2 with the node labels next to the nodes.

tribution P_T using the node labels and the functions ψ_u and ψ_{uv} as defined for G (see Equation (1.1)):

$$P_T(\mathbf{x}) = \frac{1}{Z} \left(\prod_{[y,u] \in V_T} \psi_u(x_y) \right) \left(\prod_{([y,u],[z,v]) \in E_T} \psi_{uv}(x_y, x_z) \right), \quad \mathbf{x} \in \mathcal{X}_T. \qquad (1.2)$$

In the above, analogously to Equation (1.1), we have $V_T = \{1_T, 2_T, \ldots, n_T\}$, we associate a random variable X_y with each $[y, u] \in V_T$, which takes values in $\mathcal{X}_y = \mathcal{X}_u$, and we define $\mathcal{X}_T = \mathcal{X}_{1_T} \times \mathcal{X}_{2_T} \times \ldots \times \mathcal{X}_{n_T}$.

It is well-known that if BP converges, then the MAP assignment (given by the MAP estimate of P_T) of all nodes in the computation tree that are sufficiently far away from the leaves of the tree is according to the assignment that the BP algorithm converged to (see, for example, the Periodic Assignment Lemma by Weiss [63]). Nodes that are close to the leaves do not necessarily take the assignment that BP converged to. (In the above we mean by ‘leaves’ only those leaves of the computation tree that are in the lowest level of the computation tree, not the nodes in the higher levels of the computation tree that are leaves only because the nodes that they are labeled with have degree 1 in the original graph G. For example, the u_5-labeled node at distance 2 from the CT-root in the computation tree in Figure 1.5 is not considered a leaf, while the u_3-labeled node at distance 3 from the CT-root is considered a leaf.)

Theorem 1.2.1 (Weiss [63]). *Assume that the BP algorithm converges after k_0 iterations. Each node $[x, v]$ in the computation tree $T^k(u)$ ($k \geq k_0$) that is at distance at most $k - k_0$ from the CT-root of $T^k(u)$ has MAP assignment equal to the assignment that v converged to.*

1.3 Minimum-Cost Flow Problem

In this section we introduce the *minimum-cost flow* (MCF) problem. The MCF problem consists of finding a cheapest flow that satisfies all capacity and budget constraints in a *flow network*. A flow network is a simple directed graph $G = (V, E)$

together with a *capacity function* $u\colon E \to \mathbb{R}^+$. In principle we allow multiple edges between a pair of nodes, but for ease of notation we consider simple directed graphs without directed cycles of length two. In the MCF problem there are an additional *cost function* $c\colon E \to [0,1]$ and a *budget function* $b\colon V \to \mathbb{R}$ indicating how much of a resource some node v requires ($b_v < 0$) or offers ($b_v > 0$). A *feasible b-flow* for such an instance is a function $f\colon E \to \mathbb{R}^+$ that obeys the capacity constraints $0 \le f_e \le u_e$ for any edge $e \in E$ and Kirchhoff's law adapted to the node budgets, i.e.,

$$b_v + \sum_{e=(w,v)\in E} f_e = \sum_{e'=(v,w)\in E} f_{e'},$$

for all nodes $v \in V$. (Even though b, u, c, and f are functions, we use the notation b_v, u_e, c_e, and f_e instead of $b(v)$, $u(e)$, $c(e)$, and $f(e)$.) If $\sum_{v\in V} b_v \neq 0$, then there does not exist a feasible b-flow. We therefore always require $\sum_{v\in V} b_v = 0$. The costs of a feasible b-flow are defined as $c(f) = \sum_{e\in E} f_e \cdot c_e$. In the minimum-cost flow problem the goal is to find the cheapest feasible b-flow, a so-called *minimum-cost b-flow*, if one exists, and to output an error otherwise.

Note that finding a perfect minimum-weight matching (see Section 1.4.1) in a bipartite graph $G = (U \cup V, E)$ is a special case of the minimum-cost flow problem [1]. We refer to Ahuja et al [1] for more details about the MCF problem.

1.3.1 Minimum-Cost Flow Algorithms

Flow problems have gained a lot of attention in the second half of the twentieth century to model, for example, transportation and communication networks [1,26]. Plenty of algorithms have been developed over the last fifty years. The first pseudo-polynomial (running-time polynomial in the size of the instance and the numeric values in the instance, such as, for example, the maximum edge capacity or cost) algorithm for the MCF problem was the Out-of-Kilter algorithm independently proposed by Minty [40] and by Fulkerson [27]. The simplest pseudo-polynomial algorithms are the primal Cycle Canceling algorithm by Klein [35] and the dual Successive Shortest Path (SSP) algorithm by Jewell [32], Iri [31], and Busacker and Gowen [15]. By introducing a scaling technique Edmonds and Karp [23] modified the SSP algorithm to obtain the Capacity Scaling algorithm, which was the first polynomial-time (running-time polynomial in the size of the instance and polylogarithmic in the numeric values) algorithm for the MCF problem.

The first strongly polynomial (running-time polynomial in the size of the instance and independent of the numeric values) algorithms were given by Tardos [59] and by Orlin [42]. Later, Goldberg and Tarjan [29] proposed a pivot rule for the Cycle Canceling algorithm to obtain the strongly polynomial Minimum-Mean Cycle Canceling (MMCC) algorithm. The fastest known strongly polynomial algorithm up to now is the Enhanced Capacity Scaling algorithm due to Orlin [43] and has a running-time of $O(m \log(n)(m + n \log n))$, where n and m denote the number of nodes and edges, respectively. For an extensive overview of MCF algorithms we suggest the paper of

Goldberg and Tarjan [30], the paper of Vygen [62], and the book of Ahuja, Magnanti, and Orlin [1].

When we compare the performance of MCF algorithms in theory and in practice, we see that algorithms that have good worst-case bounds on their running-time are not always the ones that perform best in practice. Zadeh [66] showed that there exist instances for which both the SSP algorithm and the NS algorithm have exponential running-time. Conversely, the MMCC algorithm runs in strongly polynomial time, as shown by Goldberg and Tarjan [29]. In practice however, the relative performance of these algorithms is completely different. Kovács [37] showed in an experimental study that the SSP algorithm and the NS algorithm are much faster than the MMCC algorithm on practical instances. The NS algorithm is even the fastest algorithm of all. This discrepancy can be explained by observing that the instances for which the SSP algorithm and the NS algorithm need exponential time are very contrived and unlikely to occur in practice. To better understand the differences between worst-case and practical performance for the SSP algorithm, the MMCC algorithm, and the NS algorithm we analyze these algorithms in the framework of smoothed analysis in Chapters 4 and 5.

In Chapter 4 we prove upper bounds for the running-time of the algorithms and in Chapter 5 we prove lower bounds. For the SSP algorithm we show an upper bound of $O(mn\phi)$ for the expected number of augmentation steps that it requires in the smoothed setting. This polynomial smoothed upper bound is in sharp contrast to the exponential number of augmentation steps that the SSP algorithm needs in the worst case. We also show an almost tight lower bound of $\Omega(m \cdot \min\{n, \phi\} \cdot \phi)$ for the number of augmentation steps that the SSP algorithm requires. The smoothed upper bound for the SSP algorithm is joint work with Tobias Brunsch and Heiko Röglin from the University of Bonn and appeared before in the PhD thesis by Tobias Brunsch [10]. The smoothed lower bound is by Clemens Rösner, also from the University of Bonn, and appeared before in his MSc thesis [49]. We include it here for the sake of completeness. For the MMCC algorithm we show an upper bound of $O\big(mn(n\log(n) + \log(\phi))\big)$ on the expected number of iterations that it requires in the smoothed setting. For dense graphs, this is an improvement over the $\Theta(m^2 n)$ iterations that the MMCC algorithm needs in the worst case. We also show a lower bound of $\Omega\big(m\log(\phi)\big)$ for the smoothed number of iterations that the MMCC algorithm requires. For $\phi = \Omega(n^2)$, we improve our lower bound to $\Omega(mn)$. For the NS algorithm, we show a lower bound of $\Omega(m \cdot \min\{n, \phi\} \cdot \phi)$ iterations that it requires in the smoothed setting.

For an introduction to the SSP algorithm, the MMCC algorithm, and the NS algorithm, we refer to Sections 4.2.1, 4.3.1, and 5.3.1, respectively.

1.3.2 Residual Network

Many MCF algorithms use the concept of a *residual network*, which we introduce in this section. For a pair $e = (u, v)$, we denote by e^{-1} the pair (v, u). Let G be a flow network, let c be a cost function, and let f be a flow. The *residual network* G_f is the directed graph with node set V, arc set $E' = E_\text{f} \cup E_\text{b}$, where

$$E_\text{f} = \big\{e : \; e \in E \text{ and } f_e < u_e\big\}$$

is the set of so-called *forward arcs* and

$$E_\text{b} = \big\{e^{-1} : \; e \in E \text{ and } f_e > 0\big\}$$

is the set of so-called *backward arcs*, a capacity function $u' \colon E' \to \mathbb{R}$, defined by

$$u'_e = \begin{cases} u_e - f_e & \text{if } e \in E\,, \\ f_{e^{-1}} & \text{if } e^{-1} \in E\,, \end{cases}$$

and a cost function $c' \colon E' \to \mathbb{R}$, defined by

$$c'_e = \begin{cases} c_e & \text{if } e \in E\,, \\ -c_{e^{-1}} & \text{if } e^{-1} \in E\,. \end{cases}$$

The capacity u'_e of an arc e in the residual network is also called the *residual capacity* of e. To distinguish between edges in the original network and edges in the residual network, we refer to edges in the original network as 'edges' and to edges in the residual network as 'arcs' in this thesis.

We call any flow network G' a *possible residual network* (of G) if there is a flow f for G such that $G' = G_f$. Paths and cycles in possible residual networks are called *possible paths* and *possible cycles*, respectively.

1.4 Other Combinatorial Optimization Problems

In this section we introduce the maximum-weight matching problem, the maximum-weight independent set problem, and the minimum spanning tree problem. These three problems are among the most well-known and well-studied combinatorial optimization problems, and have lots of applications in practice. We keep our introduction of the problems short, focusing mostly on the aspects that we need in the rest of this thesis. For a more elaborate introduction to all three problems we refer to Schrijver [54].

1.4.1 Maximum-Weight Matching

In this section we introduce the *maximum-weight matching* (MWM) problem. Consider an undirected weighted graph $G = (V, E)$ with $V = \{v_1, \ldots, v_n\}$, and $E \subseteq \big\{\{v_i, v_j\} = e_{ij} \mid 1 \le i < j \le n\big\}$. Each edge e_{ij} has weight $w_{ij} \in \mathbb{R}^+$. A collection of edges $M \subseteq E$ is called a *matching* if each node in V is incident to at most one edge in M. If each node is incident to exactly one edge in M, the matching M is called a *perfect matching.* We define the weight of a matching M by

$$w(M) = \sum_{e_{ij} \in M} w_{ij}.$$

The maximum-weight matching $M^\star$ of G is defined as

$$M^\star = \operatorname{argmax}\{w(M) \mid M \text{ is a matching of } G\}.$$

The bipartite maximum-weight matching problem is defined analogously. The only difference is that for this problem it is required that the graph G is bipartite, i.e., its node set V can be partitioned in two sets V_1 and V_2 such that all edges in its edge set E have exactly one endpoint in V_1 and exactly one endpoint in V_2.

A *b-matching* $M \subseteq E$ in an arbitrary graph $G = (V, E)$ is a set of edges such that every node $v_i \in V$ is incident to at most b_i edges from M, where $b_i \geq 0$. A b-matching is called perfect if every node $v_i \in V$ is incident to exactly b_i edges from M. The weight of a b-matching M is defined analogously to the weight of a matching.

1.4.2 Maximum-Weight Independent Set

In this section we introduce the *maximum-weight independent set* (MWIS) problem. Let $G = (V, E)$ be an undirected weighted graph. An *independent set* S is a subset $S \subseteq V$ of nodes such that for every edge $\{u, v\} \in E$ at most one of u and v is in S. The MWIS problem consists of finding an independent set of maximum weight. A subset of nodes $S^\star \subseteq V$ is an *MWIS* of G if and only if

$$S^\star \in \operatorname{argmax}\{w(S) \mid S \text{ is an independent set of } G\}.$$

It is straightforward to formulate the MWIS problem as an integer program by identifying each node $u \in V$ with a binary variable $x_u \in \{0, 1\}$. Here $x_u = 0$ can be interpreted as x not being part of the independent set S, while $x_u = 1$ can be interpreted as x being part of S. The integer program contains constraints that prevent two neighboring nodes from both being included in S. The integer program (IP-MWIS) is as follows

$$\begin{aligned} &\max \sum_{u \in V} w(u) x_u \\ \text{s.t.}\quad & x_u + x_v \leq 1 \qquad \text{for all } \{u, v\} \in E, \\ & x_u \in \{0, 1\}. \end{aligned}$$

We obtain the LP relaxation of IP-MWIS by relaxing the constraint that the variables x_u should take an integer value. We denote this LP relaxation by LP-MWIS.

$$\begin{aligned} &\max \sum_{u \in V} w(u) x_u \\ \text{s.t.}\quad & x_u + x_v \leq 1 \qquad \text{for all } \{u, v\} \in E, \\ & 0 \leq x_u \leq 1. \end{aligned}$$

The independent set polytope is given by all feasible solutions of LP-MWIS. It is well-known that every extreme point of the independent set polytope has $x_u \in \{0, \frac{1}{2}, 1\}$ for all $u \in V$.

1.4.3 Minimum Spanning Tree

In this section we define the *minimum spanning tree* (MST) problem. Let $G = (V, E)$ be an undirected graph. A *spanning tree* T of G is a connected subgraph $T = (V, F)$ of G, such that each node in V is incident to at least one of the edges in F and T is a tree. That is, T does not contain any cycles. The MST problem consists of finding a spanning tree of G of minimum total weight. A tree $T^\star$ is an *MST* of G if and only if

$$T^\star \in \operatorname{argmin}\{w(T) \mid T \text{ is a spanning tree of } G\}.$$

The MST problem can be solved in polynomial time using, for example, one of the greedy algorithms by Prim and Kruskal (see, for example, Schrijver [54]).

1.5 Thesis Outline

The main content of this thesis can be divided into two parts. In the first part (Chapters 2 and 3) we analyze the belief propagation algorithm applied to several combinatorial optimization problems. In the second part (Chapters 4 and 5) we analyze the running-time of three minimum-cost flow algorithms in the setting of smoothed analysis.

In Chapter 2 we analyze the BP algorithm applied to the maximum-weight matching (MWM) and minimum-cost flow (MCF) problems in the setting of smoothed analysis. Bayati et al. [5] and Gamarnik et al. [28] have shown that the BP algorithm requires a pseudo-polynomial number of iterations in the worst case when applied to the MWM and MCF problems. We show that for both problems, in the smoothed setting, the BP algorithm requires only a polynomial number of iterations with high probability. In addition, we provide lower bound instances that show that the smoothed number of iterations that the BP algorithm requires is not finite. The results from this chapter are published as [11].

In Chapter 3 we consider the BP algorithm applied to the maximum-weight independent set and minimum spanning tree problems. For both problems, we show that for most instances BP does not work well. For the BP algorithm applied to the MWIS problem we characterize exactly for which input graphs BP is guaranteed to work well. We show that BP applied to the MWIS problem converges to the optimal solution for all possible node weights when the input graph contains no odd cycles and at most one even cycle. If the input graph contains an odd cycle or at least two even cycles, there exist weights such that the BP algorithm does not converge to the optimal solution. For the BP algorithm applied to the MST problem we construct a simple input graph G for which the BP algorithm does not converge to the optimal solution. The graph G is a tree plus one additional edge. Since BP is guaranteed to converge to the optimal solution for trees, G is one of the simplest non-trivial instances. We believe that the properties of G that ensure that the BP algorithm does not converge to the optimal solution are shared by many other instances, and that BP does not converge to the optimal solution for most instances of the MST problem encountered in practice. The results from this chapter appear in [18].

In Chapters 4 and 5 we analyze several MCF algorithms in the setting of smoothed analysis. We consider the successive shortest path (SSP) algorithm, the minimum-mean cycle canceling (MMCC) algorithm, and the network simplex (NS) algorithm. Our results are grouped such that the upper bounds on the smoothed running-time of these algorithms appear in Chapter 4 and the lower bounds in Chapter 5. For the SSP algorithm we show that is has polynomial smoothed running-time, in sharp contrast with its exponential running-time in the worst case. We also show an almost tight lower bound on the smoothed running-time of the SSP algorithm. For the MMCC algorithm we show a smoothed running-time that is an improvement over the worst-case running-time for dense graphs. In addition, we show lower bounds for the smoothed running-time of the MMCC algorithm. Finally, we show a lower bound on the smoothed running-time of the network simplex algorithm. The instance that we use to show our lower bound for the NS algorithm is based on the instance that we use to show our lower bound for the SSP algorithm. The results in Chapters 4 and 5 are published as [12] and [19].

Publications underlying this thesis:

[11] Tobias Brunsch, Kamiel Cornelissen, Bodo Manthey, and Heiko Röglin. Smoothed analysis of belief propagation for minimum-cost flow and matching. *Journal of Graph Algorithms and Applications*, 17(6):647–670, 2013. Preliminary version presented at the *7th International Workshop on Algorithms and Computation (WALCOM 2013)*.

[12] Tobias Brunsch, Kamiel Cornelissen, Bodo Manthey, Heiko Röglin, and Clemens Rösner. Smoothed analysis of the successive shortest path algorithm. *SIAM Journal on Computing*, 44(6):1798–1819, 2015. Preliminary version presented at the *24th ACM-SIAM Symposium on Discrete Algorithms (SODA 2013)*.

[18] Kamiel Cornelissen and Bodo Manthey. Belief propagation for the maximum-weight independent set and minimum spanning tree problems. Submitted, 2015.

[19] Kamiel Cornelissen and Bodo Manthey. Smoothed analysis of the minimum-mean cycle canceling algorithm and the network simplex algorithm. In Dachuan Xu, Donglei Du, and Dingzhu Du, editors, *Proceedings of the 21st International Computing and Combinatorics Conference (COCOON 2015)*, volume 9198 of *Lecture Notes in Computer Science*, pages 701–712. Springer, 2015. Invited to appear in *Algorithmica*. Full version available at `http://arxiv.org/abs/1504.08251`.

CHAPTER 2

Smoothed Analysis of BP for Matching and Minimum-Cost Flow

2.1 Introduction

In this chapter we analyze the BP algorithms for computing maximum-weight matchings and minimum-cost flows in the setting of smoothed analysis. We prove upper and lower tail bounds for the number of iterations that the BP algorithms for the MWM and MCF problems need to converge to the optimal solution in the smoothed setting.

2.1.1 Previous Results

Bayati et al. [5] have proposed a variant of the BP algorithm for the maximum-weight matching problem (see Section 1.4.1), which we denote with BP-MWM. We introduce BP-MWM in short in Section 2.2.1. For a more elaborate introduction we refer to the original work [5]. Bayati et al. [5] have shown that BP-MWM correctly computes the maximum-weight matching in bipartite graphs if the MWM is unique. Convergence of BP-MWM takes pseudo-polynomial time and depends linearly on both the weight of the heaviest edge and $1/\delta$, where δ is the difference in weight between the best and second-best matching.

Belief propagation has also been applied to finding maximum-weight perfect matchings in arbitrary graphs and to finding maximum-weight b-matchings [3, 51]. For arbitrary graphs, BP-MWM does not necessarily converge [51]. However, Bayati et al. [3] and Sanghavi et al. [51] have shown that BP-MWM converges to the optimal matching if the relaxation of the corresponding linear program has an optimal solution that is unique and integral. The number of iterations needed until convergence depends again linearly on the reciprocal of the parameter δ. Bayati et al. [3] have also shown that the same result holds for the problem of finding maximum-weight b-matchings that do not need to be perfect.

Gamarnik et al. [28] have shown that BP can be used to find a minimum-cost flow, provided that the instance has a unique optimal solution. We denote their algorithm by BP-MCF and introduce it in short in Section 2.2.2. The number of iterations until convergence of BP-MCF is pseudo-polynomial and depends linearly on the reciprocal of the difference in cost between the best and second-best integer flow. In addition, they have proved a discrete isolation lemma [28, Theorem 8.1] that shows that the edge costs can be slightly randomly perturbed to ensure that, with a probability of at least 1/2, the perturbed MCF instance has a unique optimal solution. Using this result, they constructed a fully polynomial-time randomized approximation scheme (FPRAS) for solving the MCF problem using BP.

2.1.2 Our Model

We refer to Sections 1.4.1 and 1.3 for definitions of the MWM and MCF problems, respectively. We slightly deviate from the definition of the MCF problem in Section 1.3 by requiring the capacities and budgets to be integer. Note that we could also have allowed rational capacities and budgets. Since the values of the budgets and capacities do not appear in the bounds that we prove, our results are not affected by scaling all capacities and budgets by the least common denominator.

The graph and (for MCF) the capacities of the edges and the budgets of the nodes are adversarial. The costs or weights of the edges are random according to the one-step model introduced by Beier and Vöcking [8] (see Section 1.1).

We consider the general probabilistic model described below.

- The adversary specifies the graph $G = (V, E)$ and, in the case of minimum-cost flow, the integer capacities of the edges and the integer budgets (both are not required to be polynomially bounded). Additionally, the adversary specifies a probability density function $g_e : [0, 1] \to [0, \phi]$ for every edge e.

- The costs (for minimum-cost flow) or weights (for matching) of the edges are then drawn independently according to their respective density functions.

2.1.3 Our Results

We prove upper and lower tail bounds for the number of iterations that BP needs to solve maximum-weight matching problems and minimum-cost flow problems. Our upper bounds match our lower bounds up to a small polynomial factor. While previous bounds on the worst-case running-time for BP-MWM and BP-MCF are pseudo-polynomial [3, 5, 28, 51], we show that in the framework of smoothed analysis with high probability BP only requires a polynomial number of iterations to converge to the correct solution. This suggests that for instances encountered in practice, BP is unlikely to require a superpolynomial number of iterations. In the following, n and m are the number of nodes and edges of the input graph, respectively. In summary, we prove the following results:

- For the minimum-cost flow problem, the probability that BP-MCF needs more than k iterations to converge to the correct solution is upper bounded by $O(n^2 m\phi/k)$ (Sections 2.3.2 and 2.4.2). There are smoothed instances for which the probability that BP-MCF needs more than k iterations is lower bounded by $\Omega(n\phi/k)$ (Section 2.5.3).

- For the maximum-weight matching problem, the probability that BP-MWM needs more than k iterations to converge to the correct solution is upper bounded by $O(nm\phi/k)$ (Sections 2.3.1 and 2.4.1). There are smoothed instances for which the probability that BP-MWM needs more than k iterations is lower bounded by $\Omega(n\phi/k)$ (Section 2.5.3).

The upper bound for matching problems holds for the variants of BP for the maximum-weight matching problem in bipartite graphs [5] as well as for the maximum-weight (perfect) b-matching problem in general graphs [3, 51]. For the latter it is required that the polytope corresponding to the relaxation of the matching LP is integral.

To prove the upper tail bound for BP-MCF, we use a continuous isolation lemma that is similar to the discrete isolation lemma by Gamarnik et al. [28, Theorem 8.1]. We need the continuous version since we do not only want to have a unique optimal solution, but we also need to quantify the gap between the best and the second-best solutions.

Though our upper tail bounds show that with high probability BP needs only a polynomial number of iterations, our bounds are not strong enough to yield any bound on the expected number of iterations. Indeed, using the lower bound of $\Omega(n\phi/k)$ for the probability that k iterations are insufficient to find a maximum-weight matching in bipartite graphs, we show that this expectation is not finite. The lower bound even holds in the average case, i.e., if $\phi = 1$ (Section 2.5.2), and it carries over to the variants of BP for the minimum-cost flow problem and the minimum/maximum-weight (perfect) b-matching problem in general graphs mentioned above [3, 5, 28, 51]. The lower bound matches the upper bound up to a factor of $O(m)$ for matching and up to a factor of $O(nm)$ for minimum-cost flow (Section 2.5.3). The smoothed lower bound even holds for complete (i.e., non-adversarial) bipartite graphs.

Finally, let us remark that, for the minimum-cost flow problem, we bound only the number of iterations that BP-MCF needs until convergence. The messages might be super-polynomially long. However, for all matching problems, the length of the messages is always bounded by a small polynomial.

2.2 Description of the BP Algorithms

In this section we introduce in short the BP algorithm for bipartite maximum-weight matching (BP-MWM) used by Bayati et al. [5]. We also provide a short description of the BP algorithm for minimum-cost flow (BP-MCF) by Gamarnik et al. [28]. For the details of BP-MWM, BP-MCF, and the versions of BP for the (perfect) maximum-weight b-matching problem, we refer to the original papers [3, 28, 51]. Note that in order to understand the results and proofs that follow in the rest of this chapter, it is not necessary to know the details of the BP algorithms. When necessary, we discuss the differences between the various versions of BP in Sections 2.4 and 2.5.

2.2.1 BP for Maximum-Weight Matching

The BP-MWM algorithm used by Bayati et al. [5] is an iterative message-passing algorithm for computing maximum-weight matchings (MWMs). Bayati et al. define their algorithm for complete bipartite graphs $G = (U \cup V, E)$ with $U = \{u_1, u_2, \ldots, u_n\}$ and $V = \{v_1, v_2, \ldots, v_n\}$. The weight of edge $\{u_i, v_j\}$ is denoted by w_{ij}. With each node u_i they associate a random variable X_i that takes values in the set V of all neighbors of u_i. Similarly, with each node v_j they associate a random variable Y_j that takes values in the set U of all neighbors of v_j. Furthermore, they introduce compatibility functions $\psi_{u_i}(v_j) = e^{w_{ij}}$, $\psi_{v_j}(u_i) = e^{w_{ij}}$, and pairwise compatibility functions $\psi_{u_i v_j}(v_p, u_q)$. The pairwise compatibility functions enforce that the random variables describe a proper matching. They are defined as

$$\psi_{u_i v_j}(v_p, u_q) = \begin{cases} 0 & \text{if } p = j \text{ and } q \neq i, \\ 0 & \text{if } p \neq j \text{ and } q = i, \text{ and} \\ 1 & \text{otherwise.} \end{cases}$$

Let the probability distribution P_{MWM} be given by

$$P_{\mathrm{MWM}}(\mathbf{x}, \mathbf{y}) = \frac{1}{Z} \left(\prod_{u_i \in U} \psi_{u_i}(x_i) \right) \left(\prod_{v_j \in V} \psi_{v_j}(y_j) \right) \left(\prod_{\{u_i, v_j\} \in E} \psi_{u_i v_j}(x_i, y_j) \right),$$
$$x_i \in V,\ y_j \in U.$$

Note that the MAP estimate (see Section 1.2) of probability distribution P_{MWM} corresponds to the maximum-weight perfect matching on G. The reason for this is that the pairwise compatibility functions ensure that only matchings have non-zero probability and the single-variable compatibility functions ensure that the matching of maximum weight has the greatest probability. Also, P_{MWM} has the same form as probability distribution $\hat{P}$ in Equation (1.1). Therefore, the graph G and probability distribution P_{MWM} together form a pairwise MRF and we can use the BP algorithm to compute the MAP estimate. Note that the restriction to bipartite graphs is not necessary to model the MWM problem as a graphical model, but we choose to stay close to the original model by Bayati et al. [5].

BP-MWM is defined analogously to the general BP algorithm, which we defined in Section 1.2. In each iteration k, each node u_i sends a message vector $M^k_{i \to j}$ to each of its neighbors v_j. The messages can be interpreted as how “likely” the sending node thinks it is that the receiving node should be matched to a particular node in the MWM. Similarly, each node v_j sends a message vector $M^k_{i \leftarrow j}$ to each of its neighbors u_i. At the end of each iteration k, all nodes u_i and v_j compute their beliefs. These beliefs can be interpreted as the “likelihood” that a node should be matched to a particular neighbor. The greater the value of $b^k_{u_i}(j)$, the more likely it is that node u_i should be matched to node v_j. The beliefs can be used to estimate the MWM. We denote the estimated MWM in iteration k by $\hat{M}^k$. The estimated matching $\hat{M}^k$ matches each node u_i to node v_j, where $j = \mathrm{argmax}_{1 \le r \le n}\{b^k_{u_i}(r)\}$. Note that $\hat{M}^k$ does not always define a matching, since multiple nodes may be matched to the same node. However, Bayati et al. [5] have shown that if the MWM is unique, then $\hat{M}^k$ is a matching and equal to the MWM for sufficiently large k.

2.2.2 BP for Minimum-Cost Flow

The BP-MCF algorithm for minimum-cost flow uses the same idea of iterative message-passing as the BP-MWM algorithm for bipartite matching. However, the messages sent between edges and their endpoints in the minimum-cost flow version are piecewise linear convex functions instead of vectors. These functions represent estimates of the costs of sending a certain amount of flow along the edge, taking into account the cost of the edge, the capacity of the edge, the fact that flow has to be conserved at the endpoints of the edge, and the messages received from neighboring edges in the previous iteration. At the end of each iteration k, each edge e uses its belief to estimate the optimal amount x_e^k of flow on itself. Though in the first several iterations these estimates do not necessarily describe a flow, Gamarnik et al [28] have shown that after a sufficient number of iterations BP-MCF converges, and the estimates describe the minimum-cost flow. For a detailed description of BP-MCF we refer to the original work [28].

2.3 Isolation Lemmas for Matching and Minimum-Cost Flow

Before we turn to proving the upper tail bounds for the number of iterations of the BP algorithm in Section 2.4, we take a closer look at the quantity δ, which we defined above as the difference in weight or cost between the best and second-best matching or integer flow, respectively. The previous results discussed in Section 2.1.1 indicate that in order for the BP algorithm to be efficient, δ must not be too small. While δ can be arbitrarily small for weights or costs that are chosen by an adversary, it is a well-known phenomenon that δ is with high probability not too small when the weights or costs are drawn randomly. In this section we show two isolation lemmas, bounding the probability that δ is small, both for the matching and minimum-cost flow problem.

2.3.1 Maximum-Weight Matching

Beier and Vöcking [8, Section 2.1] have considered a general scenario in which an arbitrary set $S \subseteq \{0,1\}^m$ of feasible solutions is given and to every $x = (x_1, \dots, x_m) \in S$ a weight $w \cdot x = w_1x_1 + \dots + w_mx_m$ is assigned by a linear objective function. As in our model, they assume that every coefficient w_i is drawn independently according to an adversarial density function $g_i : [0,1] \to [0,\phi]$ and they define δ as the difference in weight between the best and the second-best feasible solution from S, i.e., $\delta = w \cdot x^\star - w \cdot x'$ where $x^\star = \operatorname{argmax}_{x \in S} w \cdot x$ and $x' = \operatorname{argmax}_{x \in S \setminus \{x^\star\}} w \cdot x$. They prove a strong isolation lemma that, regardless of the adversarial choices of S and the density functions g_i, the probability of the event $\delta \leq \varepsilon$ is bounded from above by $2\varepsilon\phi m$ for any $\varepsilon \geq 0$.

If we choose S as the set of incidence vectors of all matchings or (perfect) b-matchings in a given graph, Beier and Vöcking's results yield for every $\varepsilon \geq 0$ an upper bound on the probability that the difference δ in weight between the best and second-best matching or the best and second-best (perfect) b-matching is at most ε. Combined with the results in Section 2.1.1, this can immediately be used to obtain an upper tail bound on the number of iterations of the BP algorithm for these problems. We prove this upper tail bound in Section 2.4.1.

2.3.2 Minimum-Cost Flow

The situation for the minimum-cost flow problem is significantly more difficult because the set S of feasible integer flows cannot naturally be expressed with binary variables. If one introduces a variable for each edge corresponding to the flow on that edge, then $S \subseteq \{0, 1, 2, \ldots, u_{\max}\}^m$ where $u_{\max} = \max_{e \in E} u_e$. Röglin and Vöcking [48] have extended the isolation lemma to the setting of integer, instead of binary, vectors. However, their result is not strong enough for our purposes as it bounds the probability of the event $\delta \leq \varepsilon$ by $\varepsilon\phi m(u_{\max}+1)^2$ from above for any $\varepsilon \geq 0$. As this bound depends on $u_{\max}$ it would only lead to a pseudo-polynomial upper tail bound on the number of iterations of BP-MCF when combined with the results of Gamarnik et al. [28]. Our goal is, however, to obtain a polynomial tail bound that does not depend on the capacities. In the remainder of this section, we prove that the isolation lemma by Röglin and Vöcking [48] can be significantly strengthened when structural properties of the minimum-cost flow problem are exploited.

As all capacities and budgets are integers, there is always a minimum-cost flow that is integral. An additional property of our probabilistic model is that with probability 1 there do not exist two different integer flows with exactly the same costs. This follows directly from the fact that all costs are continuous random variables. Hence, without loss of generality we restrict our presentation in the following to the situation that the minimum-cost flow is unique.

In fact, Gamarnik et al. [28] have not used δ, the difference in cost between the best and second-best integer flow, to bound the number of iterations needed for BP-MCF to find the unique optimal solution of MCF, but they have used another quantity Δ. They have defined Δ as the length of the cheapest cycle in the residual network of the minimum-cost flow $f^\star$. Note that Δ is always non-negative. Otherwise, we could send one unit of flow along a cheapest cycle. This would result in a feasible integral flow with lower costs. With the same argument we can argue that Δ must be at least as large as δ because sending one unit of flow along a cheapest cycle results in a feasible integral flow different from $f^\star$ whose costs exceed the costs of $f^\star$ by exactly Δ. Hence any lower bound for δ is also a lower bound for Δ and so it suffices for our purposes to bound the probability of the event $\delta \leq \varepsilon$ from above.

The isolation lemma we prove is based on ideas that Gamarnik et al. [28, Theorem 8.1] have developed to prove that the optimal solution of a minimum-cost flow problem is unique with high probability if the costs are randomly drawn integers from a sufficiently large set. We provide a continuous counterpart of this lemma, where we bound the probability that the second-best integer flow is close in cost to the optimal integer flow.

Lemma 2.3.1. *The probability that the costs of the optimal and the second-best integer flow differ by at most $\varepsilon \geq 0$ is bounded from above by $2\varepsilon\phi m$.*

Proof. We apply the principle of deferred decisions. Consider any fixed edge e, and let the costs of all other edges be fixed by an adversary. The cost c_e of e is drawn according to its probability distribution, whose density is bounded by ϕ.

For $e \in E$, let $\underline{\mathsf{E}}^e_\varepsilon$ be the event that there exist two different integer flows $f^\star$ and f' with the following properties:

(i) $f^\star$ is optimal.

(ii) $c \cdot f^\star$ and $c \cdot f'$ differ by at most ε, i.e., $c \cdot f' \leq c \cdot f^\star + \varepsilon$.

(iii) $f^\star_e = 0$ and $f'_e > 0$.

In the above c is the vector of edge costs and $f^\star$ and f' are the vectors corresponding to the respective flows. Let $\overline{\mathsf{E}}^e_\varepsilon$ be analogously defined, except for Condition (iii) being replaced by $f^\star_e = u_e$ and $f'_e < u_e$.

Claim 2.3.2. *Let $e \in E$ be arbitrary. Assume that all costs except for c_e are fixed. Let $I \subseteq [0,1]$ be the set of real numbers such that $I = \{c_e \mid \underline{\mathsf{E}}^e_\varepsilon\}$. Then I is a subset of an interval of length at most ε.*

Proof. If $I \neq \emptyset$, let $\alpha = \min(I)$ and let $f^\star$ be an optimal integer flow for $c_e = \alpha$ with $f^\star_e = 0$. Due to the choice of α it is clear that $I \subseteq [\alpha, \infty)$ We claim that $I \subseteq [\alpha, \alpha + \varepsilon]$. If $c_e = \alpha + \eta$ for some $\eta > 0$, then $f^\star$ stays optimal, and, for any feasible integer solution f with $f_e > 0$, we have

$$\begin{aligned} c \cdot f = \sum_{\hat{e} \neq e} c_{\hat{e}} f_{\hat{e}} + (\alpha + \eta) f_e &\geq \sum_{\hat{e} \neq e} c_{\hat{e}} f_{\hat{e}} + \alpha f_e + \eta && \text{as } f_e \geq 1 \\ &\geq c \cdot f^\star + \eta && \text{as } f^\star_e = 0 \text{ and } f^\star \text{ is optimal.} \end{aligned}$$

Thus, for $\eta > \varepsilon$, the event $\underline{\mathsf{E}}^e_\varepsilon$ does not occur. □

The proof of the following claim is omitted as it is completely analogous to the proof of the previous claim.

Claim 2.3.3. *Let $e \in E$ be arbitrary. Assume that all costs except for c_e are fixed. Let $I \subseteq [0,1]$ be the set of real numbers such that $I = \{c_e \mid \overline{\mathsf{E}}^e_\varepsilon\}$. Then I is a subset of an interval of length at most ε.*

The following claim shows that, provided no event $\underline{\mathsf{E}}^e_\varepsilon$ or $\overline{\mathsf{E}}^e_\varepsilon$ occurs, the second-best integer flow is more expensive than the best integer flow by at least an amount of ε.

Claim 2.3.4. *Assume that for every edge $e \in E$ neither $\underline{\mathsf{E}}^e_\varepsilon$ nor $\overline{\mathsf{E}}^e_\varepsilon$ occurs. Let $f^\star$ be a minimum-cost flow and let $f' \neq f^\star$ be a minimum-cost integer flow that differs from $f^\star$, i.e., a second-best integer flow. Then $c \cdot f' \geq c \cdot f^\star + \varepsilon$.*

Proof. We prove the claim by contradiction. Assume to the contrary that for every edge $e \in E$ neither $\underline{\mathsf{E}}^e_\varepsilon$ nor $\overline{\mathsf{E}}^e_\varepsilon$ occurs, but that $f^\star$ and f' differ less than ε in cost. First, we prove that under our assumption that the minimum-cost flow is unique some edge e exists such that $f^\star_e \in \{0, u_e\}$ and $f'_e \neq f^\star_e$. Suppose that no such edge e exists and let $d = f^\star - f'$. Then $d_e > 0$ only if $f^\star_e < u_e$ (otherwise event $\overline{\mathsf{E}}^e_\varepsilon$ occurs) and $d_e < 0$ only if $f^\star_e > 0$ (otherwise event $\underline{\mathsf{E}}^e_\varepsilon$ occurs). From this, we can conclude that there exists a $\lambda > 0$ such that $f^\star + \lambda d$ is a feasible flow. Let $\lambda_0 = \max\{\lambda \mid f^\star + \lambda d \text{ is feasible}\}$ and $\check{f} = f^\star + \lambda_0 d$. From the assumption that the minimum-cost flow is unique it follows that $c \cdot d = c \cdot f^\star - c \cdot f' < 0$. Hence, $c \cdot \check{f} < c \cdot f^\star$, contradicting the choice of $f^\star$ as minimum-cost flow.

This argument shows that there always exists an edge e such that $f^\star_e \in \{0, u_e\}$ and $f'_e \neq f^\star_e$. As none of the events $\underline{\mathsf{E}}^e_\varepsilon$ and $\overline{\mathsf{E}}^e_\varepsilon$ occurs for this edge e, it follows that $c \cdot f' \geq c \cdot f^\star + \varepsilon$, contradicting our assumption at the start of the proof that $f^\star$ and f' differ less than ε in cost. □

From Claims 2.3.2 and 2.3.3, we obtain $\mathbb{P}(\underline{\mathsf{E}}^e_\varepsilon) \leq \varepsilon\phi$ and $\mathbb{P}(\overline{\mathsf{E}}^e_\varepsilon) \leq \varepsilon\phi$: We fix all edge costs except for c_e and then $\underline{\mathsf{E}}^e_\varepsilon$ can only occur if c_e falls into an interval of length at most ε. Since the density function of c_e is bounded from above by ϕ, this happens with a probability of at most $\varepsilon\phi$. The same holds for any $\overline{\mathsf{E}}^e_\varepsilon$. Since for Claim 2.3.4 we need that none of the events $\underline{\mathsf{E}}^e_\varepsilon$ and $\overline{\mathsf{E}}^e_\varepsilon$ occur, the lemma follows by a union bound over all $2m$ events $\underline{\mathsf{E}}^e_\varepsilon$ and $\overline{\mathsf{E}}^e_\varepsilon$. □

The isolation lemma (Lemma 2.3.1) together with the discussion about the relation between δ, the difference in cost between the best and second-best integer flow, and Δ, the length of the cheapest cycle in the residual network of the minimum-cost flow $f^\star$, immediately imply the following upper bound on the probability that Δ is small.

Corollary 2.3.5. *For any $\varepsilon > 0$, we have $\mathbb{P}(\Delta \leq \varepsilon) \leq 2\varepsilon\phi m$.*

2.4 Upper Bound on the Number of Iterations

In this section we prove an upper bound on the probability that BP needs a large number of iterations to compute maximum-weight matchings and minimum-cost flows. We show that in the smoothed analysis framework, with high probability a polynomial number of iterations is sufficient, both for MWM and MCF. This result can be interpreted as an indication that instances encountered in practice are unlikely to require a superpolynomial number of iterations.

2.4.1 Maximum-Weight Matching

We first consider the BP-MWM algorithm of Bayati et al. [5], which computes maximum-weight matchings in complete bipartite graphs G in $O(nw^*/\delta)$ iterations on all instances with a unique optimum. Here w^* denotes the weight of the heaviest edge and δ denotes the difference in weight between the best and the second-best matching. Even though it is assumed that G is a complete bipartite graph, this is not strictly necessary. If a non-complete graph is given, missing edges can just be interpreted as edges of weight 0.

With Beier and Vöcking's isolation lemma (see Section 2.3.1), we obtain the following tail bound on the number of iterations needed until convergence when computing maximum-weight perfect matchings in bipartite graphs using BP-MWM.

Theorem 2.4.1. *Let τ be the number of iterations until BP-MWM [5] converges. Then $\mathbb{P}(\tau \geq k) = O(nm\phi/k)$.*

Proof. The number of iterations until BP-MWM converges is bounded from above by $O(nw^*/\delta)$ [5]. The weight of each edge is at most 1, so $w^* \leq 1$. The upper bound exceeds k only if $\delta \leq O(n/k)$. By Beier and Vöcking's isolation lemma, we have $\mathbb{P}(\delta \leq O(n/k)) \leq O(nm\phi/k)$, which yields the bound claimed. □

This tail bound is not strong enough to yield any bound on the expected running-time of BP-MWM. But it is strong enough to show that BP-MWM has smoothed polynomial running-time with respect to the relaxed definition adapted from average-case complexity [8], where it is required that the expectation of the running-time to some power $\alpha > 0$ is at most linear. However, a bound on the expected number of iterations is impossible, and the tail bound proved above is tight up to a factor of $O(m)$ (Section 2.5).

As discussed in Section 2.1.1, BP has also been applied to finding maximum-weight (perfect) b-matchings in arbitrary graphs [3, 51]. The result is basically that BP converges to the optimal matching if the optimal solution of the relaxation of the corresponding linear program is unique and integral. The number of iterations needed until convergence depends again on "how unique" the optimal solution is. For Bayati et al.'s variant [3], the number of iterations until convergence depends on $1/\delta$, where δ is again the difference in weight between the best and the second-best matching. For Sanghavi et al.'s variant [51], the number of iterations until convergence depends on $1/c$, where c is the smallest rate by which the objective value will decrease if we move away from the optimal solution.

However, the technical problem in transferring the upper bound for bipartite graphs to arbitrary graphs is that the adversary can achieve that, with high probability or even with a probability of 1 (for larger ϕ), the optimal solution of the LP relaxation is not integral. In this case, BP does not converge. Already in the average-case, i.e., for $\phi = 1$, where the adversary has no power at all, the optimal solution of the LP relaxation has some fractional variables with high probability.

Still, we can transfer the results for bipartite matching to both algorithms for arbitrary matching if we restrict the input graphs to be bipartite, since in this case the constraint matrix of the associated LP is totally unimodular.

Theorem 2.4.2. *Let τ be the number of iterations until Bayati et al.'s [3] or Sanghavi et al.'s [51] BP for general matching, restricted to bipartite graphs as input, converges. Then $\mathbb{P}(\tau \geq k) = O(nm\phi/k)$.*

Proof. For Bayati et al.'s BP algorithm, this follows in the same way as Theorem 2.4.1 from their bound on the number of iterations until convergence, which is $O(n/\delta)$ [3, Theorem 1].

Sanghavi et al. prove that their variant of BP for general graphs converges after $O(1/\alpha)$ iterations, provided that the LP relaxation has no fractional optimal solutions. Here, α is defined as

$$\alpha = \min_{\hat{x} \neq x^\star \text{ is a vertex of } P} \frac{w \cdot (x^\star - \hat{x})}{\|x^\star - \hat{x}\|_1},$$

where $x^\star$ is the (unique) optimal solution to the relaxation and P is the matching polytope [51, Remark 2].

For any $\hat{x} \neq x^\star$, we have $\|x^\star - \hat{x}\|_1 \leq n$. Furthermore, $w \cdot (x^\star - \hat{x})$ is just the difference in weights between $x^\star$ and $\hat{x}$. Since the input graph is bipartite, all vertices of P are integral. Thus, $w \cdot (x^\star - \hat{x}) \geq \delta$, where (again) δ is the difference in weight between the best and the second-best matching. Thus, $\alpha \geq \delta/n$, which proves the theorem. □

Bayati et al. [3] and Sanghavi et al. [51] have also shown how to compute b-matchings with BP. If all b_i are even, then the unique optimum to the LP relaxation is integral. Thus, we circumvent the problem that the optimal solution might be fractional. Hence, following the same reasoning as above, the probability that BP for b-matching, where all b_i are even, runs for more than k iterations until convergence is also bounded by $O(mn\phi/k)$.

Theorem 2.4.3. *Let τ be the number of iterations until Bayati et al.'s [3] or Sanghavi et al.'s [51] BP for (perfect) b-matching, where all b_i are even, converges. Then $\mathbb{P}(\tau \geq k) = O(nm\phi/k)$.*

Proof. The theorem follows directly from Beier and Vöcking's isolation lemma and the bounds on the number of iterations of BP by Bayati et al. [3, Theorems 2 and 3] and Sanghavi et al. [51, Theorem 3], as in the proof of Theorem 2.4.2. □

2.4.2 Minimum-Cost Flow

The bound on the probability that Δ is small (Corollary 2.3.5) and the pseudo-polynomial bound by Gamarnik et al. [28] directly yield a tail bound on the number of iterations that BP-MCF needs until convergence.

Theorem 2.4.4. *Let τ be the number of iterations until BP-MCF [28] converges. Then $\mathbb{P}(\tau \geq k) = O(n^2m\phi/k)$.*

Proof. The number of iterations until BP-MCF converges is bounded from above by $\kappa Ln/\Delta$ for some constant κ, where L is the maximum cost of a simple directed path in the residual network for the optimal flow [28, Theorem 4.1]. The cost of each edge is at most 1, so $L \leq n$. The upper bound exceeds k only if $\Delta \leq \kappa n^2/k$. By Corollary 2.3.5, we have $\mathbb{P}(\Delta \leq \kappa n^2/k) \leq 2\kappa n^2 m\phi/k$, which yields the bound claimed. □

2.5 Lower Bound on the Number of Iterations

We show that the expected number of iterations necessary for the convergence of BP-MWM is unbounded. To do this, we prove a lower bound on the probability that BP-MWM needs a large number of iterations that matches the upper bound as described in Section 2.4 up to a small polynomial factor. For our lower bounds we use bipartite graphs since for non-bipartite graphs the convergence of BP-MWM is not guaranteed. Our lower bound holds even for a two-by-two complete bipartite graph with edge weights drawn independently and uniformly from the interval $[0, 1]$. In the following analysis, we consider the BP variant introduced by Bayati et al [5]. However, our results can be extended to other versions of BP for matching and minimum-cost flow [3, 28, 51] in a straightforward way. We discuss such extensions in Section 2.6.

We first discuss the average case, i.e., $\phi = 1$. We consider the average case separately since for our lower bounds in the smoothed setting we need that ϕ is sufficiently large ($\phi \geq 26$). For the average case we obtain a lower bound of $\Omega(n/k)$ for the probability that more than k iterations are needed for convergence (Section 2.5.2). For this lower bound, we use a simple adversarial graph. We leave it as an open problem whether or not the lower bound also holds for the (non-adversarial) complete bipartite graph on n nodes. After that, we consider (non-adversarial) complete bipartite graphs with smoothed weights and prove a lower bound of $\Omega(n\phi/k)$ for the probability that more than k iterations are needed for convergence (Section 2.5.3). We conclude this section with a discussion of how to extend our results to the other variants of BP for matching and minimum-cost flow (Section 2.6).

2.5.1 Computation Tree and T-matchings

For proving the lower bounds, we need the notion of a computation tree, which we defined in Section 1.2.2. We call a collection Λ of edges in the computation tree $T^k(x)$ a *T-matching* if no two edges of Λ are adjacent in $T^k(x)$ and each non-leaf node of $T^k(x)$ is the endpoint of exactly one edge from Λ. Leaves can be the endpoint of either one or zero edges from Λ. (Here 'leaves' are again only those leaves in the lowest level of the computation tree, see Section 1.2.2.) We define $t^k(v_i; v_r)$ as the weight of a maximum weight T-matching in $T^k(v_i)$ that uses the $\{v_i, v_r\}$-labeled edge at the root.

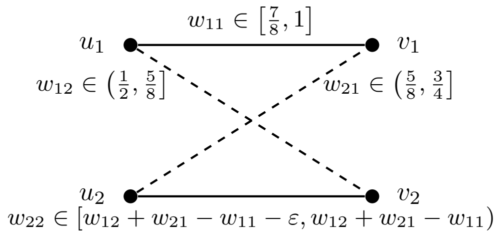

u_1
v_1
u_2
v_2
$w_{11} \in \left[\frac{7}{8}, 1\right]$
$w_{12} \in \left(\frac{1}{2}, \frac{5}{8}\right]$
$w_{21} \in \left(\frac{5}{8}, \frac{3}{4}\right]$
$w_{22} \in [w_{12} + w_{21} - w_{11} - \varepsilon, w_{12} + w_{21} - w_{11})$

Figure 2.1: If event E_ε occurs, then the weight of the dashed matching $M_2 = \{e_{12}, e_{21}\}$ is greater than the weight of the solid matching $M_1 = \{e_{11}, e_{22}\}$ and the weight difference is at most ε. In addition w_{11} is greater than w_{12} and the weight difference is at least $\frac{1}{4}$.

2.5.2 Average-Case Analysis

Consider the undirected weighted complete bipartite graph $K_{2,2} = (U \cup V, E)$, where $U = \{u_1, u_2\}$, $V = \{v_1, v_2\}$, and $\{u_i, v_j\} \in E$ for $1 \le i, j \le 2$. Each edge $\{u_i, v_j\} = e_{ij}$ has weight w_{ij} drawn independently and uniformly at random from $[0, 1]$. We define the event E_ε for $0 < \varepsilon \le \frac{1}{8}$ as the event that $w_{11} \in \left[\frac{7}{8}, 1\right]$, $w_{12} \in \left(\frac{1}{2}, \frac{5}{8}\right]$, $w_{21} \in \left(\frac{5}{8}, \frac{3}{4}\right]$, and $w_{22} \in [w_{12} + w_{21} - w_{11} - \varepsilon, w_{12} + w_{21} - w_{11})$. Consider the two possible matchings $M_1 = \{e_{11}, e_{22}\}$ and $M_2 = \{e_{12}, e_{21}\}$. If event E_ε occurs, then the weight of M_2 is greater than the weight of M_1 and the weight difference is at most ε. In addition, w_{11} is greater than w_{12} and the weight difference is at least $1/4$. See Figure 2.1 for a graphical illustration of the graph $K_{2,2}$ and the event E_ε.

Lemma 2.5.1. *The probability of event E_ε is $\varepsilon/8^3$.*

Proof. The intervals in which w_{11}, w_{12}, and w_{21} have to assume values in order for event E_ε to occur all have a length of $1/8$. The interval in which w_{22} has to take a value in order for event E_ε to occur, has a length of ε and it is contained completely in the interval $\left(0, \frac{1}{2}\right]$, since

$$w_{12} + w_{21} - w_{11} - \varepsilon > \frac{1}{2} + \frac{5}{8} - 1 - \frac{1}{8} = 0$$

and

$$w_{12} + w_{21} - w_{11} \le \frac{5}{8} + \frac{3}{4} - \frac{7}{8} = \frac{1}{2}.$$

Now the probability that w_{11}, w_{12}, w_{21}, and w_{22} all take values in the intervals necessary for event E_ε to occur is $\varepsilon/8^3$. $\square$

Lemma 2.5.2. *If event E_ε occurs, then the belief of node u_1 of $K_{2,2}$ at the end of the $4k$-th iteration is incorrect for all integers $k \le \frac{1}{8\varepsilon} - 1$.*

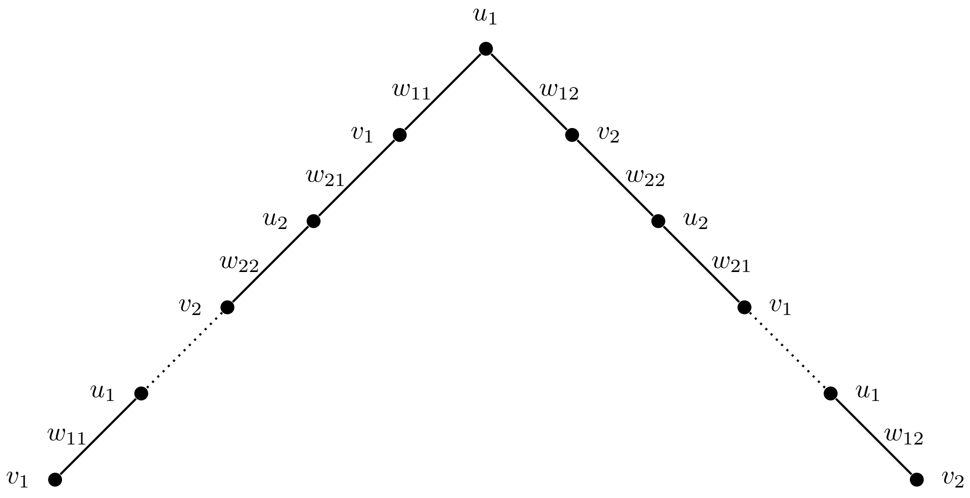


Figure 2.2: The computation tree $T^{4k}(u_1)$.

Proof. Consider the computation tree $T^{4k}(u_1)$ (see Figure 2.2). According to Bayati et al. [5, Lemma 1], the belief of node u_1 of $K_{2,2}$ after $4k$ iterations is given by the two-dimensional vector $b_{u_1}^{4k} = \big(2t^{4k}(u_1; v_1) \;\; 2t^{4k}(u_1; v_2)\big)$. This means that, after $4k$ iterations, the belief of node u_1 that it should be matched to v_1 is equal to twice the weight of the maximum-weight T-matching of $T^{4k}(u_1)$ that selects the $\{u_1, v_1\}$-labeled edge at the root. Analogously, after $4k$ iterations, the belief of node u_1 that it should be matched to v_2 is equal to twice the weight of the maximum-weight T-matching of $T^{4k}(u_1)$ that selects the $\{u_1, v_2\}$-labeled edge at the root. The maximum-weight T-matching $\hat{\Lambda}$ that matches the root node to its v_2-labeled child matches each u_1-labeled node to a v_2-labeled node and each $u-2$-labeled node to a v_1-labeled node, since this is the only possible T-matching that matches the root node to its v_2-labeled child. Define $\Lambda^\star$ as the T-matching that matches each u_1-labeled node to a v_1-labeled node and each u_2-labeled node to a v_2-labeled node. We show that $\Lambda^\star$ has larger weight than $\hat{\Lambda}$, which implies that the belief at node u_1 after $4k$ iterations is incorrect. We have

$$\begin{aligned} w(\Lambda^\star) - w(\hat{\Lambda}) &= (2k+1)w_{11} + 2kw_{22} - (2k+1)w_{12} - 2kw_{21} \\ &= 2k(w_{11} + w_{22} - w_{12} - w_{21}) + w_{11} - w_{12} \\ &\geq -2k\varepsilon + 1/4. \end{aligned}$$

Now $-2k\varepsilon + 1/4$ is greater than zero if $k \leq \frac{1}{8\varepsilon} - 1$. □

Theorem 2.5.3. *The probability that BP-MWM requires at least k iterations to converge for $K_{2,2}$ with edge weights drawn independently and uniformly from $[0,1]$ is at least $\frac{1}{\kappa k}$ for some constant $\kappa > 0$.*

Proof. We denote the number of iterations necessary for convergence of BP-MWM by τ. Using Lemma 2.5.1 and Lemma 2.5.2, we have

$$\mathbb{P}(\tau \geq k) \geq \mathbb{P}(\tau \geq 4\lceil k/4 \rceil) \geq \mathbb{P}\left(E_{\frac{1}{8(\lceil k/4 \rceil+1)}}\right)$$
$$= \frac{1}{8^4(\lceil k/4 \rceil + 1)} \geq \frac{1}{\kappa k}$$

for some constant $\kappa > 0$. □

Corollary 2.5.4. *There exist bipartite graphs on $n \geq 4$ nodes, where n is a multiple of 4, with edge weights drawn independently and uniformly from $[0,1]$, for which the probability that BP-MWM needs at least k iterations to converge is $\Omega\big(\frac{n}{k}\big)$ for $k \geq n/\kappa'$ for some constant $\kappa' > 0$.*

Proof. The bipartite graph G consists of $n/4$ copies of $K_{2,2}$ and there are no edges between nodes in different copies of $K_{2,2}$. If BP-MWM does not converge in fewer than k iterations for at least one of the $n/4$ copies of $K_{2,2}$, then BP-MWM does not converge in fewer than k iterations for G. This holds since a run of BP-MWM on G corresponds to $n/4$ parallel runs of BP-MWM on the $n/4$ copies of $K_{2,2}$. Using Theorem 2.5.3, we have that a constant $\kappa > 0$ exists such that

$$\mathbb{P}(\tau < k) = \big(1 - \mathbb{P}\big(\text{BP-MWM needs at least } k \text{ iterations to converge for a particular copy of } K_{2,2}\big)\big)^{n/4}$$
$$\leq \left(1 - \frac{1}{\kappa k}\right)^{n/4} \leq \exp\left(-\frac{n}{4\kappa k}\right) \leq 1 - \frac{n}{8\kappa k},$$

where the second inequality follows from $1 - x \leq \exp(-x)$ and the final inequality follows from $\exp(-x) \leq 1 - \frac{x}{2}$ for $x \in [0,1]$ and from $\frac{n}{4\kappa k} \leq 1$ which holds if $k \geq \frac{n}{4\kappa}$. □

2.5.3 Smoothed Analysis

In this section we consider (non-adversarial) complete bipartite graphs $K_{n,n}$ in the smoothed setting. We denote by $X \sim U[a,b]$ that random variable X is uniformly distributed on interval $[a,b]$. In the following, we assume both that $\phi \geq 26$ and $n \geq 2$ and even. Similarly to the average case (Section 2.5.2), we define the event E_ε^ϕ for $K_{2,2}$ and for $0 < \varepsilon \leq 1/\phi$ as the event that $w_{11} \in \big[1 - \frac{1}{\phi}, 1\big]$, $w_{12} \in \big(\frac{23}{26}, \frac{23}{26} + \frac{1}{\phi}\big]$, $w_{21} \in \big(\frac{23}{26}, \frac{23}{26} + \frac{1}{\phi}\big]$, and $w_{22} \in [w_{12} + w_{21} - w_{11} - \varepsilon, w_{12} + w_{21} - w_{11})$. Consider the two possible matchings $M_1 = \{e_{11}, e_{22}\}$ and $M_2 = \{e_{12}, e_{21}\}$. If event E_ε^ϕ occurs, then the weight of M_2 is greater than the weight of M_1 and the weight difference is at most ε. In addition, w_{11} is greater than w_{12} and the weight difference is at least $\frac{3}{26} - \frac{2}{\phi}$.

Lemma 2.5.5. *There exist probability distributions on $[0,1]$ for the weights of the edges, whose densities are bounded by $\phi \geq 26$, such that the probability of event E_ε^ϕ is at least $\varepsilon\phi/4$.*

Proof. The intervals in which w_{11}, w_{12}, and w_{21} have to assume values in order for event E_ε^ϕ to occur all have a length of $\frac{1}{\phi}$. We choose the corresponding probability distributions such that they have density ϕ on the corresponding interval and density 0 elsewhere. The interval in which w_{22} has to assume a value in order for event E_ε^ϕ to occur has a length of ε and it is contained completely in the interval $\left[\frac{20}{26} - \frac{1}{\phi}, \frac{20}{26} + \frac{3}{\phi}\right]$, since

$$w_{12} + w_{21} - w_{11} - \varepsilon > \frac{23}{26} + \frac{23}{26} - 1 - \frac{1}{\phi} = \frac{20}{26} - \frac{1}{\phi}$$

and

$$w_{12} + w_{21} - w_{11} \leq \left(\frac{23}{26} + \frac{1}{\phi}\right) + \left(\frac{23}{26} + \frac{1}{\phi}\right) - \left(1 - \frac{1}{\phi}\right) = \frac{20}{26} + \frac{3}{\phi}.$$

We choose the probability distribution for w_{22} such that it has density $\frac{\phi}{4}$ on the interval $\left[\frac{20}{26} - \frac{1}{\phi}, \frac{20}{26} + \frac{3}{\phi}\right]$ and 0 elsewhere. Now the probability that w_{11}, w_{12}, and w_{21} take values in the interval necessary for event E_ε^ϕ to occur is 1. For w_{22}, this probability is $\varepsilon\phi/4$. This completes the proof of Lemma 2.5.5. □

Lemma 2.5.6. *If event E_ε^ϕ ($\phi \geq 26$) occurs, then the belief of node u_1 at the end of the $4k$-th iteration is incorrect for all integers $k \leq \frac{1}{52\varepsilon} - 1$.*

Proof. As in Lemma 2.5.2, a maximum-weight T-matching that selects the $\{u_1, v_1\}$-labeled edge at the root has greater weight than a maximum-weight T-matching that selects the $\{u_1, v_2\}$-labeled edge at the root for these values of k. □

Analogously to the proof of Theorem 2.5.3, Lemmas 2.5.5 and 2.5.6 above immediately yield a lower bound of $\Omega(\phi/k)$ for the probability that BP-MWM will run for at least k iterations for $K_{2,2}$.

Our goal in the remainder of this section is to prove an $\Omega(n\phi/k)$ lower bound for the complete bipartite graph. Thus, let us consider the complete bipartite graph $K_{n,n} = (U \cup V, E)$ with $U = \left\{u_p^j \mid p \in \{1,2\}, j \in \{1,\dots,n/2\}\right\}$ and $V = \left\{v_q^j \mid q \in \{1,2\}, j \in \{1,\dots,n/2\}\right\}$. Let H^j denote the subgraph induced by $\{u_1^j, u_2^j, v_1^j, v_2^j\}$ for $j \in \{1,\dots,n/2\}$. The role of the subgraphs H^j is the same as the role of the copies of $K_{2,2}$ in the proof of Corollary 2.5.4. Let e_{pq}^j be the edge connecting u_p^j and v_q^j ($p, q \in \{1,2\}$, $j \in \{1,\dots,n/2\}$). The weight of this edge is w_{pq}^j. We draw edge weights according to the probability distributions

$$\begin{aligned}
w_{11}^j &\sim U\left[1 - \frac{1}{\phi}, 1\right], & w_{12}^j &\sim U\left(\frac{23}{26}, \frac{23}{26} + \frac{1}{\phi}\right], \\
w_{21}^j &\sim U\left(\frac{23}{26}, \frac{23}{26} + \frac{1}{\phi}\right], & w_{22}^j &\sim U\left[\frac{20}{26} - \frac{1}{\phi}, \frac{20}{26} + \frac{3}{\phi}\right], \\
w_{ab} &\sim U\left[0, \frac{1}{\phi}\right] \text{ if } u_a \in H^j \text{ and } v_b \in H^\ell \text{ with } j \neq \ell.
\end{aligned} \tag{2.1}$$

We call the edges between nodes in the same induced subgraph H^j *heavy edges.* Edges between nodes in different subgraphs H^j and H^ℓ we call *light edges.* By assumption, we have $\phi \geq 26$. Thus, the weight of any light edge is at most $1/26$, while every heavy edge weighs at least $19/26$.

In contrast to the proof of Corollary 2.5.4, we now have to make sure that light edges are not used in any computation tree. This allows us to prove the lower bound in a similar way to that in Theorem 2.5.3 and Corollary 2.5.4.

Lemma 2.5.7. *Let $\Lambda^\star$ be the maximum-weight T-matching on the computation tree $T^k(u)$. Then $\Lambda^\star$ does not contain any light edges.*

Proof. Assume to the contrary that $\Lambda^\star$ contains a light edge $\{x, y\}$. In that case, x and y are in different subgraphs. The idea of the proof is to construct a path P from one leaf of the computation tree to another leaf that includes edge $\{x, y\}$. Path P alternately consists of edges that are in $\Lambda^\star$ and edges that are not. We show that a new T-matching of greater weight can be constructed by removing from $\Lambda^\star$ the edges in $P \cap \Lambda^\star$ and adding the edges in $P \setminus \Lambda^\star$.

We include the $\{x, y\}$-labeled edge in P and extend P on both sides. We start with node $z_0 = x$ and node $z_0 = y$, respectively, and construct the corresponding part of P as follows:

```
1. for $i = 1, 3, 5, \ldots$ do
2.     if $z_{i-1}$ is a leaf node then terminate.
3.     Let $H^\ell$ be the subgraph that $z_{i-1}$ belongs to.
4.     Let $e_i = \{z_{i-1}, z_i\}$ be the edge incident to $z_{i-1}$ that belongs to the optimal
       matching with respect to $H^\ell$.
5.     Add $e_i$ to $P$.
6.     if $z_i$ is a leaf node then terminate.
7.     Let $e_{i+1} = \{z_i, z_{i+1}\}$ be the (unique) edge incident to $z_i$ that belongs to $\Lambda^\star$.
8.     Add $e_{i+1}$ to $P$.
```

It is clear that the procedure can only terminate if it finds a leaf. Moreover, the constructed sequence is alternating. Now we can show that no node will be visited twice. Otherwise, there is an index i such that $z_{i-1} = z_{i+1}$ since P is a path in a tree. However, this can not happen since the sequence is alternating. Therefore, the procedure terminates. Using the previous properties we also obtain that both paths constructed starting with $z_0 = x$ and $z_0 = y$, respectively, are disjoint since $z_1 \notin \{x, y\}$ in both cases. Consequently, we obtain one simple path P connecting two distinct leaf nodes and containing edge $\{x, y\}$.

We now show that the weight of the edges in $P \setminus \Lambda^\star$ is strictly larger than the weight of the edges in $P \cap \Lambda^\star$. For this, let P be of the form $P = (p_0, \ldots, p_t)$, where t is even and where $\{p_0, p_1\} \in \Lambda^\star$. Let $I \subseteq \{1, \ldots, t\}$ be the set of indices i for which $\{p_{i-1}, p_i\}$ is a light edge. Clearly, $\{p_{i-1}, p_i\} \in \Lambda^\star$ for each $i \in I$ by construction (see Line 4). Since the light edge $\{x, y\}$ belongs to P we have $I \neq \emptyset$. For $i \in I$, let $P_i = (p_{i-1}, p_i, p_{i+1})$ be the subpath of P of length 2 starting at node p_{i-1}. As $\{p_i, p_{i+1}\}$ is a heavy edge, $w_{p_i p_{i+1}} - w_{p_{i-1} p_i} \geq \left(\frac{20}{26} - \frac{1}{\phi}\right) - \frac{1}{\phi} = \frac{20}{26} - \frac{2}{\phi}$. Therefore, the difference in weight between the edge of P_i that belongs to $\Lambda^\star$ and the other edge is significant.

Now remove all paths P_i from P and consider the subpaths of P (connected components) that remain. There are at most $|I|+1$ such subpaths P'; each has even length, and they only consist of heavy edges, i.e., all their edges lie in one subgraph H^ℓ where ℓ depends on P'. Consider such a subpath P' and partition it into subpaths $\tilde{P}_j$ of length 4 and, if the length of P' is not a multiple of 4, into one subpath $\hat{P}$ of length 2. The $\Lambda^\star$-edges of $\tilde{P}_j$ form the non-optimal matching on H^ℓ, whereas the other two edges form the optimal matching on H^ℓ. Hence, the total weight of $\tilde{P}_j \cap \Lambda^\star$ is at most the total weight of $\tilde{P}_j \setminus \Lambda^\star$. Only for $\hat{P}$ might we have the case that the weight of $\hat{P} \cap \Lambda^\star$ is larger than the weight of $\hat{P} \setminus \Lambda^\star$, but since both edges are heavy, the difference is at most $1 - \left(\frac{20}{26} - \frac{1}{\phi}\right) = \frac{6}{26} + \frac{1}{\phi}$. Hence, the difference between the total weight of $P \setminus \Lambda^\star$ and the total weight of $P \cap \Lambda^\star$ is at least

$$|I| \cdot \left(\frac{20}{26} - \frac{2}{\phi}\right) - (|I|+1) \cdot \left(\frac{6}{26} + \frac{1}{\phi}\right) = |I| \cdot \left(\frac{14}{26} - \frac{3}{\phi}\right) - \left(\frac{6}{26} + \frac{1}{\phi}\right) \geq \frac{4}{26} > 0,$$

since $|I| \geq 1$ and $\phi \geq 26$.

We can now construct a T-matching with higher weight than $\Lambda^\star$ by removing the edges in $P \cap \Lambda^\star$ from $\Lambda^\star$ and adding the edges in $P \setminus \Lambda^\star$. This contradicts the assumption that the maximum weight T-matching includes a light edge and proves the lemma. □

Theorem 2.5.8. *There exist probability distributions on* $[0,1]$ *for the weights of the edges, whose densities are bounded by* $\phi \geq 26$*, such that the probability that BP-MWM requires at least* k *iterations to converge for* $K_{n,n}$ *is* $\Omega(n\phi/k)$ *for* $k \geq n\phi/c$ *for some constant* $c > 0$.

Proof. We choose the probability distributions for the edge weights according to (2.1). Let $\varepsilon = \frac{1}{52(k'+1)}$ for $k' = 4\lceil k/4 \rceil$ and assume that event E_ε^ϕ occurs for subgraph H^j. In this case, the weight of matching $M_2 = \{e_{12}^j, e_{21}^j\}$ is higher than the weight of matching $M_1 = \{e_{11}^j, e_{22}^j\}$, but at most by the small amount of ε. Consider the computation tree $\hat{T} = T^{4k'}(u_1^j)$. As in the proof of Lemma 2.5.2 we know that if the maximum weight T-matching $\Lambda^\star$ on $\hat{T}$ does not include the e_{12}^j-labeled edge at the root, then BP-MWM has not yet converged within the first $4k' \geq k$ iterations (see Bayati et al. [5, Lemma 1]).

We show that $\Lambda^\star$ does not include the e_{12}^j-labeled edge at the root. Assume to the contrary that it does. We know from Lemma 2.5.7 that $\Lambda^\star$ does not contain light edges. Now we use the same procedure as in Lemma 2.5.7 to create a path P from one leaf of $\hat{T}$ to another leaf, such that P contains the e_{12}^j-labeled edge at the root and alternates between edges from $\Lambda^\star$ and edges from $\hat{T} \setminus \Lambda^\star$. Since $\hat{T}$ has height $4k'+1$ and since u_1^j is the root of $\hat{T}$, path P contains exactly $8k'+2$ edges, $2k'+1$ of which are e_{12}^j-labeled, $2k'+1$ of which are e_{11}^j-labeled, $2k'$ of which are e_{22}^j-labeled, and $2k'$ of which are e_{21}^j-labeled. The e_{12}^j-labeled and e_{21}^j-labeled edges are exactly the edges of $P \cap \Lambda^\star$. As in Lemma 2.5.2, the difference of weight between edges from $P \setminus \Lambda^\star$ and $P \cap \Lambda^\star$ is at least

$$\begin{aligned} w_{11}^j - w_{12}^j - 2k'\varepsilon &\geq \left(\left(1 - \frac{1}{\phi}\right) - \left(\frac{23}{26} + \frac{1}{\phi}\right)\right) - \frac{2k'}{52(k'+1)} \\ &> \frac{3}{26} - \frac{2}{\phi} - \frac{1}{26} \geq 0 \end{aligned}$$

since $\phi \geq 26$. This contradicts the fact that $\Lambda^\star$ is optimal since removing from $\Lambda^\star$ the edges in $P \cap \Lambda^\star$ and adding the edges in $P \setminus \Lambda^\star$ yields a T-matching of heavier weight for $\hat{T}$.

We have shown that BP-MWM does *not* converge within the first k iterations if event E_ε^ϕ occurs for some subgraph H^j. Since there are $n/2$ such subgraphs, we find that the probability that BP-MWM needs at least k iterations to converge for $K_{n,n}$ is $\Omega\left(\frac{n\phi}{k}\right)$ since

$$\begin{aligned} \mathbb{P}(\tau \leq k) &\leq \mathbb{P}\left(E_\varepsilon^\phi \text{ does not occur for any subgraph } H^j\right) \\ &\leq \left(1 - \frac{\varepsilon\phi}{4}\right)^{n/2} \leq \exp\left(-\frac{\varepsilon n\phi}{8}\right) = \exp\left(-\frac{n\phi}{8 \cdot 52 \cdot (4 \cdot \lceil k/4 \rceil + 1)}\right) \\ &\leq 1 - \frac{n\phi}{2 \cdot 8 \cdot 52 \cdot (4 \cdot \lceil k/4 \rceil + 1)} \end{aligned}$$

where the second inequality follows from Lemma 2.5.5. The third inequality is due to the fact that $1-x \leq \exp(-x)$, whereas the last inequality stems from $\exp(-x) \leq 1-\frac{x}{2}$ for $x \in [0,1]$. If $x = \frac{n\phi}{8 \cdot 52 \cdot (4 \cdot \lceil k/4 \rceil + 1)}$ is at most 1, which holds for $k \geq \frac{n\phi}{8 \cdot 52}$, then the correctness follows. □

Note that the lower bound on the probability that BP-MWM converges within k iterations only differs by a factor $O(m)$ from the upper bound as proved in Section 2.4.1.

2.6 Concluding Remarks

The lower bounds presented in Sections 2.5.2 and 2.5.3 also hold for other versions of belief propagation for minimum/maximum-weight (perfect) b-matching and minimum-cost flow [3, 28, 51] applied to the matching problem on bipartite graphs. The number of iterations until convergence differs by no more than a constant factor between these versions of BP. We omit the technical details but provide some comments on how the proofs need to be adjusted.

Some of the versions of BP consider minimum-weight perfect matching [3] or minimum-cost flow [28] instead of maximum-weight perfect matching. For these versions, we obtain the same results if we have edge weights $\tilde{w}_e = 1 - w_e$ for all edges e.

For some versions of BP [28, 51], the root of the computation tree is an edge rather than a node. If we choose the root of this tree suitably, then we have that the difference in weight between the two matchings M_1 and M_2 of at most ε not only has to "compensate" the weight difference $\Delta w(e_1, e_2)$ between an edge e_1 in M_1 and an edge e_2 in M_2, but the entire weight w_e of an edge e in M_1 or M_2. However, the probability distributions for the edge weights described in Sections 2.5.2 and 2.5.3 are chosen such that $\Delta w(e_1, e_2)$ and w_e do not differ more than a constant factor.

Note that our lower bounds for the number of iterations until convergence of BP-MWM in the average case (see Section 2.5.2) do not contradict the results reported by Salez and Shah [50]. They consider complete bipartite graphs instead of the adversarial graphs that we use. Roughly speaking, Salez and Shah have proved that BP for bipartite matching requires only a constant number of iterations. However, they allow that in expectation a small constant fraction of the nodes are matched to incorrect nodes. It might even be the case that multiple nodes are matched to the same node. In our analysis, we require convergence of the BP algorithm, i.e., each node should be matched to the unique node to which it is matched in the optimal matching.

Even though the graphs we consider in Section 2.5.2 are different from the graphs Salez and Shah consider, and even though they consider upper bounds on the number of iterations of BP required and we consider lower bounds, some of our results are similar in flavor to their results. Theorem 2.5.3 only provides a constant lower bound on the number of iterations required to achieve any fixed probability of convergence of BP for a single $K_{2,2}$. By linearity of expectation, we only obtain a constant lower bound on the number of iterations required for convergence of any fixed fraction of the (independent) copies of $K_{2,2}$'s in expectation, as in the paper by Salez and Shah. On the other hand, Corollary 2.5.4 shows that a constant number of iterations is not sufficient to have convergence for all of the copies of $K_{2,2}$'s.

CHAPTER 3

BP for Independent Set and Minimum Spanning Tree

3.1 BP for Independent Set

3.1.1 Introduction

Sanghavi et al. [53] have introduced a variant of BP for the MWIS problem (see Section 1.4.2), which we denote by BP-MWIS. They have shown that BP-MWIS does not converge to the correct independent set if the LP relaxation of the problem has a non-integral optimal solution. Also, they have shown that even if the LP relaxation of the problem has a unique integral optimal solution, BP-MWIS is not guaranteed to converge. In this section we characterize precisely the graph structures for which BP-MWIS is guaranteed to work well. This means that we characterize the graph structures for which BP-MWIS is guaranteed to converge to the correct solution irrespective of the node weights, as long as the MWIS is unique. We show that the graphs for which BP-MWIS converges to the correct solution for all possible node weights are exactly those graphs that contain at most one even cycle and no odd cycles. In other words, BP-MWIS is only guaranteed to work well for bipartite graphs that are trees plus at most one additional edge.

Previous Results

BP-MWIS is a variant of the BP algorithm for the MWIS problem developed by Sanghavi et al. [53]. For a graph $G = (V, E)$ they associate with each node $u \in V$ a random variable X_u that takes values from the set $\{0, 1\}$. A value of '0' for X_u can be interpreted as u not being part of independent set S, while a value of '1' can be interpreted as u being part of S. They define $\psi_u(x_u) = e^{w(u)x_u}$, $\psi_{uv}(x_u, x_v) = 0$ if $x_u + x_v > 1$, and $\psi_{uv}(x_u, x_v) = 1$ otherwise. Let the probability distribution P_{IS} be given by

$$P_{\text{IS}}(\mathbf{x}) = \frac{1}{Z}\left(\prod_{u\in V}\psi_u(x_u)\right)\left(\prod_{(u,v)\in E}\psi_{uv}(x_u, x_v)\right), \qquad \mathbf{x} \in \{0,1\}^{|V|}.$$

For distribution P_{IS}, only $\mathbf{x}$ corresponding to independent sets of G have positive probability. Since the MAP estimate of P_{IS} corresponds to the MWIS of G, BP can be used as a heuristic for computing the MWIS of G. BP-MWIS is the BP algorithm by Sanghavi et al. for the graphical model given by graph G and probability distribution P_{IS}. In each iteration of BP-MWIS each node sends two messages $m_{u\to v}(0)$ and $m_{u\to v}(1)$ to each of the nodes v in its neighborhood $N(u)$. These messages can be interpreted as the likelihood according to the sending node u that the receiving node should not be in the MWIS ($m_{u\to v}(0)$) or should be in the MWIS ($m_{u\to v}(1)$), respectively. Since the exact structure of the messages does not play a role in our analysis, we will not further specify them and refer to the original paper. At the end of each iteration each node estimates whether it should be in the MWIS. We denote the estimate of node u in iteration k by $x_u^k \in \{0, 1, ?\}$. An estimate of '0' can be interpreted as u believing that it should not be in the MWIS, an estimate of '1' can be interpreted as u believing that it should be in the MWIS, and an estimate of '?' as u considering it equally likely that it is part of the MWIS or not.

Sanghavi et al. have shown that fixed points of BP-MWIS correspond to vertices of the polytope P described by LP relaxation LP-MWIS (see Section 1.4.2). The fixed point that BP-MWIS converges to depends on the initialization of the messages and does not necessarily correspond to the vertex of P corresponding to the optimal solution of LP-MWIS. However, if BP-MWIS is started with neutral initial messages (the setting that we consider here, see Section 1.2), then it converges to the optimal solution of LP-MWIS, if it converges at all. For more details we refer to the original work.

In our analysis we use several results by Sanghavi et al. [53] which we list below.

Theorem 3.1.1 (Sanghavi et al. [53]). *If LP-MWIS has a non-integral optimal solution, then BP-MWIS does not converge to the correct solution.*

Just like for the original graph G, we can consider maximum-weight independent sets of a computation tree $T^k(u)$ (see Section 1.2.2 for an introduction to computation trees). The estimates x_u^k of BP-MWIS can be directly related to whether or not the CT-root of the computation tree $T^k(u)$ is part of a MWIS of $T^k(u)$.

Theorem 3.1.2 (Sanghavi et al. [53]). *For any node $u \in V$ and any number of iterations k we have*

- $x_u^k =$ '1' *if and only if the CT-root of $T^k(u)$ is a member of every MWIS of $T^k(u)$;*
- $x_u^k =$ '0' *if and only if the CT-root of $T^k(u)$ is not a member of any MWIS of $T^k(u)$;*
- $x_u^k =$ '?' *otherwise.*

Our Results

In Sections 3.1.2 and 3.1.3 we characterize the graphs G for which BP-MWIS converges to the correct solution for all possible node weights (assuming that the MWIS is unique). We show that BP-MWIS is only guaranteed to work well for bipartite graphs that are trees plus at most one additional edge. In Section 3.1.2 we show that BP-MWIS converges to the correct solution for all possible node weights for all G that contain no odd length cycles and at most one even length cycle. In Section 3.1.3 we show that if G contains an odd length cycle or at least two even length cycles, there exist node weights for which BP-MWIS does not converge to the correct solution.

3.1.2 Graphs for Which BP-MWIS Converges

In this section we show that BP-MWIS converges to the correct solution for all possible node weights for graphs that contain no odd cycles and at most one even cycle.

Theorem 3.1.3. *Let $G = (V, E)$ be a graph that contains no odd cycle and at most one even cycle. Then BP-MWIS converges to the correct solution for all possible node weights w for which the MWIS of G is unique.*

Proof. If G is a tree, then after at most n iterations, the computation tree T is equal to G. Since the MWIS of G is unique, the MWIS of T is unique as well and according to Theorem 3.1.2 BP-MWIS converges to the correct solution.

Next we consider the case that G contains exactly one even cycle $C = (W, F)$ and no odd cycles. Let $q = |W|$. We denote the nodes in C by $v_0, v_1, v_2, \ldots, v_q = v_0$ such that $\{v_i, v_{i+1}\} \in F$. Furthermore, we define sets $V_1, V_2, \ldots, V_q$ where V_i consists of node v_i plus all nodes u that are not on the cycle C and for which the shortest path from u to the cycle ends in v_i. We also define weights

$$\begin{aligned} w_i^+ &= \max\{w(B) \mid v_i \in B, B \subset V_i, B \text{ is an independent set of } G\} \quad \text{and} \\ w_i^- &= \max\{w(B) \mid v_i \notin B, B \subset V_i, B \text{ is an independent set of } G\}. \end{aligned}$$

We denote by $V_i^+ \subset V_i$ and by $V_i^- \subset V_i$ the subsets for which the weights w_i^+ and w_i^- are obtained, respectively, breaking ties arbitrarily. Using the above definitions, the problem of finding the MWIS of G can be reduced to finding the independent set $D \subset W$ for which

$$\sum_{i:v_i \in D} w_i^+ + \sum_{i:v_i \notin D} w_i^- \quad \text{is maximized.}$$

We denote the MWIS of G by I. Also, we denote by I' an arbitrary second-heaviest independent set, that is

$$I' \in \operatorname{argmax}\{w(S) \mid S \subset V, S \neq I, S \text{ is an independent set of } G\}.$$

Since the MWIS of G is unique, there is a strictly positive difference between the weight of I and the weight of I'. We define $\delta = w(I) - w(I') > 0$. We denote the weight of the heaviest node of G by w^*.

Let $T = (V_T, E_T)$ be a computation tree for G and let $R \subset V_T$ be a subset of W-labeled nodes of the computation tree. In the following we denote by $M[R]$ the subgraph of T that is induced by R plus all nodes u in T that are not W-labeled and for which a path from u to some $v \in R$ exists for which all nodes except for v are not W-labeled.

Note that from the above definitions we immediately obtain

$$w_i^+ \leq w_i^- + w^*, \tag{3.1}$$

since $B = V_i^+ \setminus \{v_i\}$ is an independent set of G, $B \subset V_i$, and $v_i \notin B$. Also, we have

$$w_i^+ \geq w_i^- \text{ if } v_i \in I, \tag{3.2}$$

because otherwise, we can improve I by removing the nodes in V_i^+ and then adding the nodes in V_i^-.

We first show that BP-MWIS converges to the correct solution for nodes $v \in W \cap I$. Assume to the contrary that BP-MWIS does not converge to the correct solution for v. We define $k^* = n(\frac{nw^*}{\delta})$. Then, according to Theorem 3.1.2 there exists a $k > k^* + 3n$ such that the CT-root of the computation tree $T = T^k(v)$ is not a part of every MWIS of T. Let J be an MWIS of T that does not include the CT-root. We now define sets S^+ and S^- on T recursively. We start by adding the CT-root to S^+. Each time we add a node to S^+, we add to S^- each of its neighbors in the computation tree that is W-labeled, in J, and at distance at most $k^* + 2n + 1$ from the CT-root. Each time we add a node to S^-, we add to S^+ each of its neighbors in the computation tree that is W-labeled, I-labeled, and at distance at most $k^* + 2n$ from the CT-root.

Note that the nodes in $S^+ \cup S^-$ induce a path P that starts at a v_i-labeled node, continues to a v_{i+1}-labeled node, etc., and ends in a v_j-labeled node. We can partition this path into shorter paths, such that p parts $P_1, \ldots, P_p$ are equal to $(v_i, v_{i+1}, \ldots, v_{i-1})$, that is, every P_ℓ is equal to cycle C with edge $\{v_{i-1}, v_i\}$ removed. In addition, the partition consists of at most one part P^* of length less than $|W|$ which is equal to $(v_i, v_{i+1}, \ldots, v_j)$.

Next we show that we can construct an independent set $\tilde{J}$ on T of weight greater than $w(J)$ as follows. We set $\tilde{J} = J$. For each node $[u, v_i]$ in S^+ we first remove from $\tilde{J}$ all nodes in $M[\{u\}]$, then add to $\tilde{J}$ all V_i^+-labeled nodes in $M[\{u\}]$. In addition, for each node $[u, v_i]$ in S^- we first remove from $\tilde{J}$ all nodes in $M[\{u\}]$, then add to $\tilde{J}$ all V_i^--labeled nodes in $M[\{u\}]$. Note that $\tilde{J}$ is again an independent set on T, since the W-labeled neighbors of each node $[u, v_i] \in S^+$ are either in S^- and therefore not in $\tilde{J}$, or they are not in J (otherwise they would have been added to S^-) and therefore not in $\tilde{J}$ either.

Now we consider one path P_ℓ and the graph $M_\ell = (V_{M_\ell}, E_{M_\ell}) = M[P_\ell]$. Note that M_ℓ is a copy of G, except for the missing edge $\{v_{i-1}, v_i\}$. The set of labels of the nodes in $V_{M_\ell} \cap \tilde{J}$ is exactly equal to I. Also, the set of labels of the nodes in $V_{M_\ell} \cap J$ is equal to some other independent set $\hat{I}$ of G. Since I is at least δ heavier then any other independent set of G, we have that

$$w(V_{M_\ell} \cap \tilde{J}) \geq w(V_{M_\ell} \cap J) + \delta. \tag{3.3}$$

In the following we denote by $M^* = (V_{M^*}, E_{M^*}) = M[P^*]$. We now distinguish two cases.

Case 1: $|P| > k^* + n$**.** Since $|P| > k^* + n$, we have $p \geq k^*/n + 1$. By Equation (3.3), we have $w(V_{M_\ell} \cap \tilde{J}) \geq w(V_{M_\ell} \cap J) + \delta$ for all ℓ. By Equations (3.1) and (3.2) we have $w(V_{M^*} \cap J) \leq w(V_{M^*} \cap \tilde{J}) + (n-1)w^*$. Combining these two inequalities yields

$$w(\tilde{J}) - w(J) \geq p\delta - (n-1)w^* > 0.$$

Since $\tilde{J}$ is heavier than J, our assumption that BP-MWIS does not converge to the correct solution for node v, graph G, and weights w was incorrect.

Case 2: $|P| \leq k^* + n$**.** Let $[x, v_i]$ and $[y, v_j]$ be the endpoints of P. Suppose $x \in S^+$. Since $x \in S^+$, we have $v_i \in I$ by definition and therefore $v_{i-1} \notin I$. Suppose now $x \in S^-$. Since the v_{i-1}-labeled neighbor u of x was not added to S^+, u cannot be I-labeled by definition, so $v_{i-1} \notin I$. Similarly, the v_{j+1}-labeled neighbor of y that is not in P cannot be I-labeled, so $v_{j+1} \notin I$. Consider now P^*. Suppose that $J \cap V_{M^*}$ is at least as heavy as $\tilde{J} \cap V_{M^*}$. Since neither v_{i-1} nor v_{j+1} is in I, we can define a new independent set $\tilde{I}$ of G of weight at least $w(I)$. We set $\tilde{I} = I$. Next, we remove from $\tilde{I}$ all nodes in the sets V_i for which v_i is used to label one of the nodes in P^*. By doing so, $w(\tilde{I})$ decreases by $w(\tilde{J} \cap V_{M^*})$. Then we add to $\tilde{I}$ all nodes that are used to label one of the nodes in $J \cap V_{M^*}$. By doing so, $w(\tilde{I})$ increases by $w(J \cap V_{M^*})$. Since the nodes in $J \cap V_{M^*}$ are at least as heavy as the nodes in $\tilde{J} \cap V_{M^*}$, we have that $\tilde{I}$ is at least as heavy as I. This contradicts the fact that I is the unique MWIS of G. Therefore, our assumption that $J \cap V_{M^*}$ is at least as heavy as $\tilde{J} \cap V_{M^*}$ was wrong. Since also $\tilde{J} \cup V_{M_\ell}$ is heavier than $J \cup V_{M_\ell}$ for all ℓ, we have that $\tilde{J}$ is heavier than J.

Since $\tilde{J}$ is heavier than J, our assumption that BP-MWIS does not converge to the correct solution for node v, graph G, and weights w was incorrect.

We have shown convergence of BP-MWIS to the correct solution for nodes $v \in W \cap I$. Next we consider nodes $v \in W \backslash I$. The proof that BP-MWIS converges to the correct solution for these nodes is very similar to the proof for nodes that are in I. Assume that BP-MWIS does not converge to the correct solution for v. Then, according to Theorem 3.1.2, there exists a $k > k^* + 3n$ such that the CT-root of the computation tree $T = T^k(v)$ is part of some MWIS J on T. We now define sets S^+ and S^- analogously to the proof for $v \in I$ and start the recursive definition of these sets by including the CT-root in S^-. We can then show that J is not an MWIS of T, so the assumption that BP-MWIS does not converge to the correct solution for $v \in W, v \notin I$ was wrong. We omit the rest of the proof, since it is very similar to the proof for $v \in I$.

Finally, we show that BP-MWIS converges to the correct solution for nodes $v \notin W$. Assume w.l.o.g. that $v \in V_1$ and let d be the length of the shortest path from v to v_1 in G. Note that $T = T^{k+d}(v)$ is exactly the same as $\hat{T} = T^k(v_1)$ for $k \geq n$ (except that the CT-root is different). Since these two computation trees are the same, also the MWISs on these trees are the same. We denote the v_1-labeled node that is closest to the CT-root in T by u. Since u corresponds to the CT-root of $\hat{T}$, $v_1 \in W$, and BP-MWIS converges to the correct solution for nodes in W, node u is in every MWIS of T if $v_1 \in I$ and it is in no MWIS of T if $v_1 \notin I$. Let $M = M[\{u\}]$. We now consider the case where u is in every MWIS of T. In computation tree T, all nodes in $M \setminus \{u\}$ are only connected to other nodes in M. Therefore, every MWIS J on T with $u \in J$ includes each $[x, y] \in M$ if and only if $y \in V_1^+$. If $y \in V_1^+$ and $v_1 \in I$, then also $y \in I$. On the other hand, if $y \notin V_1^+$ and $v_1 \in I$, then also $y \notin I$. This holds in particular for the CT-root. It will be in every MWIS of T if $v \in I$ and in no MWIS of T if $v \notin I$. The case that u is in no MWIS of T is similar and we therefore omit the proof.

□

3.1.3 Graphs for Which BP-MWIS Does Not Converge

In Section 3.1.2 we have shown that BP-MWIS converges to the correct solution for all possible node weights for graphs with at most one even cycle and no odd cycles. In this section we show that these are the only graphs for which BP-MWIS converges to the correct solution for all possible node weights. First we show that there exist node weights such that BP-MWIS does not converge to the correct solution for graphs that contain an odd cycle and then we show that there exist node weights such that BP-MWIS does not converge to the correct solution for graphs that contain two or more even cycles.

In our proofs we use the concept of *heavy* nodes and *light* nodes. We denote the set of heavy nodes by H and the set of light nodes by L. The heavy nodes all have weight at least 1. We do not specify the exact weights of the light nodes, but they all have weight from the interval $]0, 1/9n^2[$ such that the weights of all subsets of L are different, that is, $w(S) = w(T) \Rightarrow S = T$ for all $S, T \subset L$. We choose the node weights like this to ensure that the MWIS is unique.

First we consider graphs with at least one odd cycle. For these graphs our result follows directly from Theorem 3.1.1 and the fact that for graphs with an odd cycle we can choose node weights such that LP-MWIS does not have an integral optimal solution.

Theorem 3.1.4. *Let $G = (V, E)$ be a graph that contains at least one odd cycle $C = (W, F)$. Then there exist weights for the nodes such that the MWIS of G is unique, but BP-MWIS does not converge to the correct solution.*

Proof. Let $q = |W|$. We denote the nodes in C by $v_0, v_1, v_2, \ldots, v_q = v_0$ such that $\{v_i, v_{i+1}\} \in F$. We choose the node weights such that the nodes in $S = \{v_1, v_3, v_5, \ldots, v_{k-4}, v_{q-2}\}$ have weight $1 + 1/(2n)$, the nodes in $W \setminus S$ have weight

1, and all nodes in $V \setminus W$ are light nodes. We show that the optimal solution of LP-MWIS is non-integral.

The MWIS of G consists of the nodes in S plus some light nodes. This is because we can include at most $(q-1)/2$ nodes from W and including a node from $W \setminus S$ instead of a node in S costs us $1/(2n)$, while we can gain at most $(n-q)(1/(9n^2)) < 1/(9n)$ by including more of the light nodes. The weight of the MWIS of G is therefore bounded by $((q-1)/2)(1+1/(2n)) + (n-q)(1/(9n^2)) < q/2$. By our assumption on the weights of the light nodes, the MWIS is unique.

Let x be the solution of LP-MWIS with $x_i = 0$ if $i \notin W$ and $x_i = 1/2$ if $i \in W$. The objective value for x is clearly greater than $q/2$. This shows that LP-MWIS cannot have an integral optimal solution and according to Theorem 3.1.1 BP-MWIS does not converge to the correct solution. □

Next we consider graphs with at least two even cycles.

Theorem 3.1.5. *Let $G = (V, E)$ be a graph that contains at least two even cycles $C_1 = (W_1, F_1)$ and $C_2 = (W_2, F_2)$. There exist node weights such that the MWIS of G is unique, but BP-MWIS does not converge to the correct solution.*

Proof. If G contains an odd cycle, then the theorem follows from Theorem 3.1.4. We therefore assume in the following that G is bipartite. We now define a set X of nodes and a set Y of edges as follows. If C_1 and C_2 have at least one node in common, we define $X = W_1 \cup W_2$ and $Y = F_1 \cup F_2$. If C_1 and C_2 have no nodes in common, let $P = (W_P, F_P)$ be an arbitrary path from W_1 to W_2. In this case we define $X = W_1 \cup W_2 \cup W_P$ and $Y = F_1 \cup F_2 \cup F_P$. Note that all nodes in X have degree at least 2 in the graph $M = (X, Y)$, since either they are on one of the two cycles, or they are a non-leaf node of the path P.

Since M is a connected bipartite graph, we can uniquely partition the nodes in X into two sets X_1 and X_2 such that there are no edges $\{x_1, x_2\}$ in Y between a node $x_1 \in X_1$ and an node $x_2 \in X_2$. We now distinguish two cases.

Case 1: $|X_1| \neq |X_2|$**.** Assume w.l.o.g. that $|X_1| > |X_2|$. We define weights $\tilde{w}$ for the nodes in X as follows. Each node $x_1 \in X_1$ has weight $\tilde{w}(x_1) = 1$ and each node $x_2 \in X_2$ has weight $\tilde{w}(x_2) = 1 + 1/(2n)$. Let $S \subset X$ be an arbitrary MWIS of M. Since G is bipartite, S is an independent set of G as well. We now define weights w on the nodes V of G as follows. All nodes in $V \setminus X$ are light nodes. Each node $x \in S$ has weight $w(x) = \tilde{w}(x) + 1/(4n)$. Finally, each node $x \in X \setminus S$ has weight $w(x) = \tilde{w}(x)$. By choosing the weights w like this, we ensure that the MWIS J of G is unique and consists of the nodes in S plus the heaviest subset $\hat{L}$ of light nodes such that nodes in $\hat{L}$ are not incident to nodes in S or other nodes in $\hat{L}$. The MWIS is unique, since the nodes in S have total weight at least $1/(4n)$ greater than any other subset of X and the total weight of all light nodes is at most $1/(9n)$.

Note that at least one of the nodes in X_1 is part of J, since X_1 is an independent set of M and it has total weight greater than any subset of nodes $D \subset X_2$, because of $|X_1| > |X_2|$. Let $x_1 \in X_1$ be part of J. Assume that BP-MWIS converges to the correct solution in k_0 iterations. We consider the computation tree $T = T^k(x_1)$ for some even $k \geq k_0$. Since BP-MWIS converges to the correct solution by assumption and because of Theorem 3.1.2, the CT-root of T is a member of every MWIS of T. We now show by induction that this is not the case and that our assumption that BP-MWIS converges to the correct solution is wrong. In particular, we show that all X_1-labeled nodes are in no MWIS of T, while all X_2-labeled nodes are in every MWIS of T. Note that a node u in T that is heavier than all of its neighbors together is in every MWIS of T, since we can always improve independent sets of T that do not include u by including u and removing all neighbors of u.

As the basis step, we consider the leaves of T. Since the leaves are at an odd distance from the CT-root, they cannot be X_1-labeled. If they are X_2-labeled, they are in every MWIS of T, since they have greater weight than their parent node.

As the induction step, we consider the nodes at distance t from the CT-root. We assume that for all nodes at distance greater than t from the CT-root it holds that they are part of no MWIS of T if they are X_1-labeled and that they are part of every MWIS of T if they are X_2-labeled. For even t, nodes cannot be X_2-labeled. X_1-labeled nodes u at distance t from the CT-root have at least one X_2-labeled neighbor v which is at distance $t+1$ from the CT-root. Since v is part of every MWIS of T by assumption, u is part of no MWIS of T. For odd t, nodes cannot be X_1-labeled. An X_2-labeled node u at distance t from the CT-root is in every MWIS of T, since its X_1-labeled neighbors at distance $t+1$ from the CT-root are in no MWIS of T by assumption and its parent plus its light neighbors in T have total weight less than $w(u)$.

Case 2: $|X_1| = |X_2|$. The only connected graphs for which all nodes have degree at most 2 are paths and cycles. Since M is connected and is neither a path nor a cycle, it must contain at least one node with degree at least 3. Assume w.l.o.g. that node $x \in X_1$ has degree at least 3. We define weights $\tilde{w}$ for the nodes in X as follows. Node x has weight $\tilde{w}(x) = 5/3$. Each node $x_1 \in X_1 \setminus x$ has weight $\tilde{w}(x_1) = 1$ and each node $x_2 \in X_2$ has weight $\tilde{w}(x_2) = 1 + 1/(2n)$. Let $S \subset X$ be an arbitrary MWIS of M. We now define weights w on the nodes V of G as follows. All nodes in $V \setminus X$ are light nodes. Each node $x \in S$ has weight $w(x) = \tilde{w}(x) + 1/(4n)$. Finally, each node $x \in X \setminus S$ has weight $w(x) = \tilde{w}(x)$. Again, this way we ensure that the MWIS J of G is unique and consists of the nodes in S plus the heaviest subset $\hat{L}$ of light nodes such that nodes in $\hat{L}$ are not incident to nodes in S or other nodes in $\hat{L}$. The MWIS is unique, since the nodes in S have total weight at least $1/(4n)$ greater than any other independent set on M and the total weight of all light nodes is at most $1/(9n)$.

Note that at least one of the nodes in X_1 is part of J, since X_1 is an independent set on M and it has total weight greater than any subset of nodes $D \subset X_2$. Let $x_1 \in X_1$ be part of J. Assume that BP-MWIS converges to the correct solution in k_0 iterations. We consider the computation tree $T = T^k(x_1)$ for some even $k \geq k_0$. Since BP-MWIS converges to the correct solution and by Theorem 3.1.2, the CT-root of T is a member of every MWIS of T. We now show by induction that this is not the case and that our assumption that BP-MWIS converges to the correct solution is wrong. In particular, we show that all X_1-labeled nodes are in no MWIS of T, while all X_2-labeled nodes are in every MWIS of T.

As the basis step, we consider the leaves of T. Since the leaves are at an odd distance from the CT-root, they cannot be X_1-labeled. The X_2-labeled leaves that do not have an x-labeled node as their parent are in every MWIS of T, since they have greater weight than their parent node. Now consider an X_2-labeled leaf u that has an x-labeled node v as its parent. Since x has degree at least 3 in M, v has at least two heavy leaves as its children. Therefore, v cannot be in any MWIS of T, since we can improve any independent set of T containing v by removing v and adding its children in T. Since v is the only neighbor of u and v is in no MWIS of T, node u is in every MWIS of T.

As the induction step, we consider the nodes at distance t from the CT-root. We assume that for all nodes at distance greater than t from the CT-root it holds that they are part of no MWIS of T if they are X_1-labeled and that they are part of every MWIS of T if they are X_2-labeled. For even t, nodes cannot be X_2-labeled. X_1-labeled nodes u at distance t from the CT-root have at least one X_2-labeled neighbor v which is at distance $t+1$ from the CT-root. Since v is part of every MWIS of T by assumption, u is part of no MWIS of T. For odd t, nodes cannot be X_1-labeled. For X_2-labeled nodes u at distance t from the CT-root we again distinguish two cases. If the parent of u is not x-labeled, u is in every MWIS of T, since its X_1-labeled neighbors at distance $t+1$ from the CT-root are in no MWIS of T by assumption and its parent plus its light neighbors in T have total weight less than $w(u)$. If the parent v of u is x-labeled, it has at least two heavy X_2-labeled nodes as its children. Node v cannot be in any MWIS of T, since we can improve any independent set of T containing v by adding all its heavy children C in T and removing all neighbors of nodes in C (since X_1-labeled neighbors at distance greater than t from the CT-root are in no MWIS this always leads to an improvement), leading to a contradiction. Since v is in no MWIS of T and all X_1-labeled neighbors of u at distance $t+1$ from the CT-root are in no MWIS of T by assumption, u is in every MWIS of T. □

3.2 BP for Minimum Spanning Tree

3.2.1 Introduction

Bayati et al. [4] introduced a variant of the BP algorithm for the MST problem (see Section 1.4.3), which we denote by BP-MST. The MST problem is easily solvable in polynomial time using a variety of algorithms (see, for example, Schrijver [54]). Still, it is interesting to analyze the performance of the BP algorithm applied to the MST problem, since the MST problem has a global connectivity constraint. This is in contrast to, for example, the MWM, MCF, and MWIS problems, which only have local constraints. Therefore, a message-passing algorithm like BP seems unsuitable to solve the MST problem. Bayati et al. have shown the following positive result for BP-MST: if BP-MST converges, then it converges to the correct solution. In contrast, we provide a negative result for BP-MST. In Section 3.2.2 we show a small instance for which BP-MST does not converge. In addition, the property of this instance that ensures that BP-MST does not converge is quite general and carries over to many other instances. Therefore, we believe that BP-MST does not converge for most instances encountered in practice.

Previous Results

BP-MST is a variant of the BP algorithm for the MST problem developed by Bayati et al. [4]. We give a short description of their algorithm below. Also, we state their results that we use in Section 3.2.2 to show that BP-MST does not converge for all instances of the MST problem. For a more elaborate description of the algorithm we refer to the original paper.

Bayati et al. model a spanning tree of an undirected graph $G = (V, E)$ as a rooted directed tree. One of the nodes in V is designated as the root of the tree. To distinguish the root of a rooted spanning tree from the root of a computation tree, we call the former the MST-root. Each node $u \in V$ has an associated parent node $p_u \in N(u)$ and an associated depth $d_u \in \{0, 1, \dots, n-1\}$. (Though Bayati et al. [4] did not specify the value $d_{\max}$ for the maximum depth of a node, we make the natural choice of $d_{\max} = n - 1$.) The MST-root has (by definition) itself as its parent and depth 0. For each other node u it has to hold that $\{u, p_u\} \in E$ and that $d_{p_u} = d_u - 1$. Note that every spanning tree of G can be modeled in this way and that each set $\{(p_u, d_u)_{u \in V}\}$ that satisfies the above conditions provides a spanning tree of G. For an example of an undirected spanning tree modeled as a directed spanning tree, we refer to the right image of Figure 3.1.

In each iteration of BP-MST each node u sends a message $m_{u \to v}(p_v, d_v)$ to each of the nodes v in its neighborhood $N(u)$ for all the possible combinations of values for p_v and d_v. Such a message $m_{u \to v}(p_v, d_v)$ can be interpreted as the likelihood according to the sending node u that the receiving node v should have parent p_v and depth d_v in the MST of G. Since the exact structure of the messages does not play a role in our analysis, we will not further specify them and refer to the original paper. At the end of each iteration, each node u uses the incoming messages to estimate its parent p_u and depth d_u in the MST. Bayati et al. have shown that if BP-MST converges, it finds the MST.

Theorem 3.2.1 (Bayati et al. [4]). *If BP-MST converges to $(p_u, d_u)_{u \in V}$, then the set of edges $\big\{\{u, p_u\}_{u \in V \setminus \{\text{MST-root}\}}\big\}$ is the minimum spanning tree of G.*

For another result by Bayati et al. that we use in our analysis, we need the notion of an *Oriented Spanning Tree* (OST) on the computation tree $T^k(u)$ (see Section 1.2.2) for BP-MST. We assign to each node $[x, v]$ in $T^k(u)$ a depth $d_x \in \{0, 1, \dots, n-1\}$. To each non-leaf node $[y, v]$ in $T^k(u)$ we assign a parent p_y in its neighborhood $N([y, v])$ (or $[y, v]$ itself in case v is the MST-root of G). Here 'leaves' are again only those leaves in the lowest level of the computation tree, see also Section 1.2.2. We call such an assignment valid if it satisfies two properties:

- Every non-leaf node $[y, v]$ of $T^k(u)$ for which v is the MST-root of G has itself as its parent and depth $d_y = 0$.
- For every non-leaf node $[y, v]$ of $T^k(u)$ for which v is not the MST-root of G, it has to hold that $d_{p_y} = d_y - 1$.

Every such valid assignment gives an OST

$$\big\{([y,v],p_y)\ |[y,v] \text{ is not a leaf in the lowest level of } T^k(u) \text{ and } v \text{ is not the MST-root of } G\big\}.$$

Among all OSTs on the computation tree, we call the one of minimum weight the *Minimum-Weight Oriented Spanning Tree* (MWOST) and we call the problem of computing it the MWOST problem. Bayati et al. have shown that BP-MST solves the MWOST problem on the computation tree.

Theorem 3.2.2 (Bayati et al. [4]). *BP-MST solves the MWOST problem on the computation tree. That is, the MAP assignment of all nodes in the computation tree is such that it corresponds to the MWOST on the computation tree. In particular, for all $u \in V$, the estimates p_u^k and d_u^k at the end of iteration k are equal to the values of $p_{\text{CT-root}}$ and $d_{\text{CT-root}}$ in the MWOST of $T^k(u)$.*

Our Results

Theorem 3.2.2 shows that BP-MST actually computes the MWOST of the computation tree and not the MST of G. Nevertheless, according to Theorem 3.2.1, if BP-MST converges, then it finds the MST of G. However, convergence of BP-MST is not guaranteed. In Section 3.2.2 we show a small example graph G for which BP-MST does not converge, and we explain why we believe that BP-MST does not converge for most graphs encountered in practice.

3.2.2 Non-Convergence of BP-MST

In this section we provide an instance of the MST problem for which BP-MST does not converge to the correct solution. The instance $G = (V, E)$ is as follows (see Figure 3.1):

- $V = \{u_1, u_2, u_3, u_4, u_5, u_6, u_7\}$;
- $E = \big\{\{u_1,u_2\},\{u_1,u_3\},\{u_1,u_4\},\{u_1,u_5\},\{u_5,u_6\},\{u_5,u_7\},\{u_6,u_7\}\big\}$;
- The weights of the edges are $w(u_1, u_5) = 2$, $w(u_5, u_6) = 1$, and the rest of the edges have weight 0.

As can easily be observed, the MST $T^\star$ of G consists of all edges except for the edge $\{u_5, u_6\}$. Modeled as a directed spanning tree rooted at u_1, the set S of parents and depths corresponding to $T^\star$ is given by

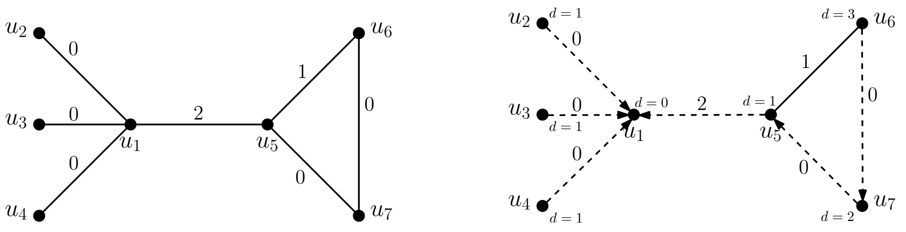


Figure 3.1: The left image shows the instance for which BP-MST does not converge. The right image shows the MST (dashed edges) for this instance, modeled as a directed tree rooted at u_1.

$$\begin{aligned} S = \{&(p_{u_1} = u_1, d_{u_1} = 0), (p_{u_2} = u_1, d_{u_2} = 1), (p_{u_3} = u_1, d_{u_3} = 1),\\ &(p_{u_4} = u_1, d_{u_4} = 1), (p_{u_5} = u_1, d_{u_5} = 1), (p_{u_6} = u_7, d_{u_6} = 3),\\ &(p_{u_7} = u_5, d_{u_7} = 2)\}. \end{aligned} \tag{3.4}$$

Before we formally prove that BP-MST for G does not converge to $T^\star$, we give an intuitive explanation of why this is the case. Note that in any spanning tree of G the expensive edge $\{u_1, u_5\}$ has to be included, since this is the only edge that connects the nodes $\{u_1, u_2, u_3, u_4\}$ with the nodes $\{u_5, u_6, u_7\}$. However, copies of the edge $\{u_1, u_5\}$ in the computation tree are not necessarily included in an oriented spanning tree (OST). In fact, for any u_5-labeled node in the computation tree it is cheaper to have either its u_6-labeled neighbor or its u_7-labeled neighbor as its parent than its u_1-labeled neighbor.

We show that BP-MST does not converge for G by proof by contradiction. Assume to the contrary that BP-MST for G converges. According to Theorems 1.2.1, 3.2.1, and 3.2.2, if we consider a sufficiently large computation tree T, the MWOST $\hat{T}$ on T should consist of copies of $T^\star$ close to the root of T. Therefore, $\hat{T}$ contains several (u_5, u_1)-labeled edges. We show that we can construct an OST on T with lower costs than $\hat{T}$ by replacing an (expensive) (u_5, u_1)-labeled edge by a (cheaper) (u_5, u_6)-labeled edge, changing the node depths where necessary. This contradicts the optimality of $\hat{T}$. We conclude that BP-MST does not converge for G.

We proceed with the formal proof.

Lemma 3.2.3. *If BP-MST converges for G, then it converges to the set S (see Equation* (3.4)*).*

Proof. Assume that BP-MST converges for G after k_0 iterations. First we show that BP-MST converges to the correct parents p_v as given by S. For u_1 this is clear, since it is the MST-root and its parent is u_1 by definition. Now assume that for some nodes BP-MST does not converge to the correct parents. Among all these nodes, let v be one of minimum depth d_v^S as given by S and let p_v^S be the parent of v as given by S. Since S is a rooted spanning tree, p_v^S has smaller depth as given by S than v and, therefore, BP-MST converges to the correct parent for p_v^S. This means that we have neither $p_{p_v^S} = v$, nor $p_v = p_v^S$. Therefore, the edge $\{v, p_v^S\}$ is not in the set $\big\{\{v, p_v\}_{v \in V \setminus \{\text{MST-root}\}}\big\}$, contradicting Theorem 3.2.1. We conclude that BP-MST converges to the correct parents for all nodes.

Finally, we show that BP-MST converges to the correct depths d_v. For u_1 this is again true by definition. Assume that for some nodes, BP-MST converges to the incorrect depths. Among all these nodes, let v be one of minimum depth d_v^S as given by S and let p_v^S be the parent of v as given by S. Consider the computation tree $T^{k_0+1}(v)$. According to Theorem 1.2.1 and the above, the neighbor $[x, p_v^S]$ of $[\text{CT-root}, v]$ has depth $d_x = d_{p_v^S}^S$. Since $p_{\text{CT-root}} = x$ and v takes the incorrect depth, we have $d_{\text{CT-root}} = d_v \neq d_v^S = d_{p_v^S}^S + 1 = d_x + 1$, contradicting Theorem 3.2.2. We conclude that BP-MST converges to the correct depths for all nodes. □

Theorem 3.2.4. *BP-MST does not converge for G.*

Proof. Assume to the contrary that BP-MST converges for G after k_0 iterations. According to Lemma 3.2.3, BP-MST converges to the set S. We now consider the computation tree $T = T^{k_0+4}(u_5)$. According to Theorem 1.2.1, all nodes in T that are at distance at most 4 from the CT-root $[\text{root}, u_5]$ take MAP assignment according to S. We denote the OST that corresponds to the MAP assignment on T by T_1. We now show that we can change the parents and depths for some nodes in T such that we obtain another OST T_2 of weight less than the weight of T_1. Consider all nodes in T at distance at most 4 from the CT-root and all edges between them (see Figure 3.2). We make the following changes to the assignments of the nodes. We change p_{root} to $[x_2, u_6]$. We change d_{root} to 4, d_{x_3} to 5, and d_{x_8} to 6. Note that the new assignment is valid. For the nodes at distance 4 or less from the CT-root this can easily be checked and for nodes further away from the CT-root it follows since we did not change their parents and depths, and also the parents and depths of nodes at distance exactly 4 from the CT-root were not changed.

The new assignment corresponds to another OST T_2. The new tree T_2 contains exactly the same edges as T_1, except that it contains an extra (u_5, u_6)-labeled edge and it does not contain one of the (u_5, u_1)-labeled edges (see Figure 3.2). Since edge $\{u_5, u_6\}$ weighs less than edge $\{u_5, u_1\}$, T_2 weighs less than T_1. Therefore, BP-MST did not compute the MWOST on T, contradicting Theorem 3.2.2. We conclude that our initial assumption was incorrect and that BP-MST does not converge for G. □

The graph G shows that BP-MST does not converge for all graphs. Since computing the MST on a tree is trivial, G is one of the simplest non-trivial instances. Bayati et al. [4] have shown that BP-MST is correct if it converges. However, the graph G shows that there exist simple instances for which BP-MST does not converge. We believe that BP-MST does not converge for most instances encountered in practice. The reason for this is that to form the MWOST of the computation tree it is often not optimal to use copies of the MST of the input graph H. Even if the MST of H contains only one somewhat expensive edge e, an OST on the computation tree

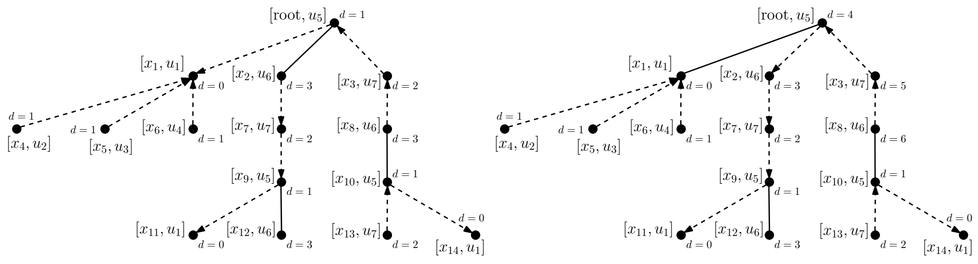


Figure 3.2: Both images show all nodes in the computation tree $T^{k_0+4}(u_5)$ that are at distance at most 4 from the CT-root $[\text{root}, u_5]$, and all edges between these nodes. The left image shows (dashed edges) the OST T_1 and the right image shows the OST T_2.

consisting of copies of the MST of H can usually be improved by leaving out a copy of edge e and adding a cheaper edge, like we showed for graph G.

3.3 Concluding Remarks

In contrast to BP-MWM and BP-MCF, which we analyzed in Chapter 2, we did not analyze BP-MWIS and BP-MST in the smoothed setting. The reason for this is that both BP-MWIS and BP-MST do not work well for many instances, and that adding a small amount of noise to the node or edge weights does not solve this. For example, consider a cycle consisting of three nodes which all have weight 1. The optimal solution to LP-MWIS for this instance is fractional, and therefore BP-MWIS does not converge to the correct solution. Even when we add a small amount of noise to the node weights, the optimal solution of LP-MWIS remains fractional, and BP-MWIS still does not converge to the correct solution. Now, consider the graph G for which we showed that BP-MST does not converge. When we add a small amount of noise to the weights of the edges of G, tree $T^\star$ remains optimal, the proof of Theorem 3.2.4 goes through, and BP-MST still does not converge.

CHAPTER 4

Smoothed Upper Bounds for Minimum-Cost Flow Algorithms

In this chapter we show upper bounds for the running-time of the successive shortest path (SSP) algorithm and the minimum-mean cycle canceling (MMCC) algorithm in the setting of smoothed analysis. The smoothed upper bound for the SSP algorithm is joint work with Tobias Brunsch and Heiko Röglin from the University of Bonn and appeared before in the PhD thesis by Tobias Brunsch [10]. Before we start our analysis, we first introduce the model that we use for the smoothed instances of the minimum-cost flow problem in Section 4.1. We also introduce the SSP algorithm in Section 4.2.1 and the MMCC algorithm in Section 4.3.1. In addition to the upper bounds, we show lower bounds for the smoothed running-time of the SSP and MMCC algorithm. These can be found in Chapter 5, where we also show a lower bound on the smoothed running-time of the network simplex algorithm. In this chapter and the next all logarithms are to the base 2, unless stated otherwise.

4.1 Our Model

For the definition of the minimum-cost flow problem we refer to Section 1.3. We use the following model of smoothed analysis. As in worst-case analysis, the network graph, the edge capacities, and the node budgets are chosen adversarially. The edge costs are random according to the one-step model introduced by Beier and Vöcking [8] (see Section 1.1).

We consider the general probabilistic model described below.

- The adversary specifies the graph $G = (V, E)$, the edge capacities and the node budgets. Additionally, the adversary specifies a probability density function $g_e : [0, 1] \to [0, \phi]$ for every edge e.

- The edge costs are then drawn independently according to their respective density functions.

4.2 Successive Shortest Path Algorithm

4.2.1 Introduction

The successive shortest path (SSP) algorithm is an algorithm that solves the MCF problem. The idea behind the algorithm is to repeatedly send flow along shortest paths from a node with positive budget to a node with negative budget, until the total outflow minus the total inflow equals the budget for every node. Since the SSP algorithm is very intuitive and easy to implement, it is very popular in practice. However, in the worst case the SSP algorithm has an exponential running-time, as shown by Zadeh [66] (see also Section 1.3.1). In this section we show that in the framework of smoothed analysis the SSP algorithm has polynomial expected running-time, in sharp contrast to its exponential worst-case running-time. Before we prove our upper bound on the smoothed running-time of the SSP algorithm, we first give a definition of the algorithm. For a more elaborate introduction to the SSP algorithm we refer to Ahuja et al. [1].

Definition of the SSP Algorithm

In practice, the simplest way to implement the SSP algorithm is to transform the instance to an equivalent instance with only one *supply node* (a node with positive budget) and one *demand node* (a node with negative budget). For this, we add two nodes s and t to the network which we call *master source* and *master sink*, edges (s, v) for any supply node v, and edges (w, t) for any demand node w. The capacities of these *auxiliary edges* (s, v) and (w, t) are set to $b_v > 0$ and $-b_w > 0$, respectively. The costs of the auxiliary edges are set to 0. Now we set $b_s = -b_t = z$ where z is the sum of the capacities of the auxiliary edges incident with s (which is equal to the sum of the capacities of the auxiliary edges incident with t due to the assumption that $\sum_{v\in V} b_v = 0$ (see Section 1.3)). All other node budgets are set to 0.

This is a well-known transformation of an arbitrary minimum-cost flow instance into a minimum-cost flow instance with only a single source s, a single sink t, and $b_v = 0$ for all nodes $v \in V \setminus \{s, t\}$. Nevertheless, we cannot assume without loss of generality that the flow network we study has only a single source and a single sink. The reason is that in the probabilistic input model introduced above it is not possible to insert auxiliary edges with costs 0 because the costs of each edge are chosen according to some density function that is bounded from above by ϕ. Therefore, we consider the auxiliary edges with costs 0 explicitly and separately from the other edges in our analysis.

The SSP algorithm run on the transformed instance computes the minimum-cost b-flow for the original instance. In the remainder of Section 4.2 we use the term *flow* to refer to a feasible b-flow for an arbitrary b with $b_s = -b_t$ and $b_v = 0$ for $v \notin \{s, t\}$. We will denote by $|f|$ the amount of flow shipped from s to t in flow f, i.e., $|f| = \sum_{e=(s,v)\in E} f_e - \sum_{e=(v,s)\in E} f_e$.

The SSP algorithm for a minimum-cost flow network with a single source s, a single sink t, and with $b_s = -b_t = z > 0$ is given as Algorithm 1. Here G_f is the

Algorithm 1 SSP algorithm for single-source-single-sink minimum-cost flow networks with $b_s = -b_t = z > 0$.

```
1:  start with the empty flow $f^0 = 0$
2:  for $i = 1, 2, \dots$ do
3:      if $G_{f^{i-1}}$ does not contain a (directed) $s$-$t$ path then
4:          output that there does not exist a flow with value $z$
5:      end if
6:      find a shortest $s$-$t$ path $P^i$ in $G_{f^{i-1}}$ with respect to the arc costs
7:      augment the flow as much as possible* along path $P^i$ to obtain a new flow $f^i$
8:      if $|f^i| = z$ then
9:          output $f^i$
10:     end if
11: end for
```

* Since the value $|f^i|$ of flow f^i must not exceed z and the flow f^i must obey all capacity constraints, the flow is increased by the minimum of $\min\{u_e - f_e^{i-1} \mid e \in P^i \cap E\}$, $\min\{f_{e^{-1}}^{i-1} \mid e \in P^i \text{ and } e^{-1} \in E\}$ and $z - |f^{i-1}|$.

residual network for flow f.

Theorem 4.2.1. *In any round i, flow f^i is a minimum-cost b^i-flow for the budget function b^i defined by $b_s^i = -b_t^i = |f^i|$ and $b_v^i = 0$ for $v \notin \{s, t\}$.*

Theorem 4.2.1 is due to Jewell [32], Iri [31], and Busacker and Gowen [15]. We refer to Korte and Vygen [36] for a proof. As a consequence, no residual network G_{f^i} contains a directed cycle with negative total costs. Otherwise, we could augment along such a cycle to obtain a b^i-flow f' with smaller costs than f^i. In particular, this implies that the shortest paths in G_{f^i} from s to nodes $v \in V$ form a shortest path tree rooted at s. Since the choice of the value z only influences the last augmentation of the algorithm, the algorithm performs the same augmentations when run for two different values $z_1 < z_2$ until the flow value $|f^i|$ exceeds z_1. We will exploit this observation in Lemma 4.2.8.

Note that one could allow the cost function c to have negative values as well. As long as the network does not contain a cycle with negative total costs, the SSP algorithm is still applicable. However, as we cannot ensure this property if the edge costs are random variables, we made the assumption that all edge costs are non-negative.

Our results

We show that the SSP algorithm has polynomial smoothed running-time. In Section 4.2.4 we prove the following upper bound on the running-time of the SSP algorithm in the setting of smoothed analysis.

Theorem 4.2.2. *The SSP algorithm requires $O(mn\phi)$ augmentation steps in expectation and its smoothed running-time is $O(mn\phi(m + n \log n))$.*

Even though the SSP algorithm has exponential running-time in the worst case, Theorem 4.2.2 suggests that one is unlikely to encounter instances for which the SSP algorithm requires an exponential amount of time. In Chapter 5 we show an almost tight lower bound of $\Omega(m \cdot \min\{n, \phi\} \cdot \phi)$ for the number of augmentation steps that the SSP algorithm requires in the smoothed setting.

In Section 4.2.5 we point out some connections between our smoothed analysis of the SSP algorithm and the smoothed analysis of the simplex method with the shadow vertex pivot rule by Spielman and Teng [56].

A Connection to the Integer Worst-case Bound

We can concentrate on counting the number of augmenting steps of the SSP algorithm since each step can be implemented to run in time $O(m + n \log n)$ using Dijkstra's algorithm (see, for example, Korte and Vygen [36]). Let us first consider the case that all edge costs are integers from $\{1, \ldots, c^*\}$. In this case the length of any path in any residual network is bounded by nc^*. We will see that the lengths of the augmenting paths are monotonically increasing. If there is no unique shortest path to augment flow along and ties are broken by choosing one with the fewest number of arcs, then the number of successive augmenting paths with the same length is bounded by $O(mn)$ (this follows from the analysis of the Edmonds-Karp algorithm for computing a maximum flow [17]). Hence, the SSP algorithm terminates within $O(mn^2c^*)$ augmentation steps.

Now let us perturb the edge costs of such an integral instance independently by, for example, uniform additive noise from the interval $[-1, 1]$. This scenario is not covered by bounds for the integral case. Nevertheless, an immediate consequence of Theorem 4.2.2 is that, in expectation, the SSP algorithm terminates within $O(mnc^*)$ augmentation steps on instances of this form.

4.2.2 Terminology and Notation

Consider the run of the SSP algorithm on the flow network G. We denote the set $\{f^0, f^1, \ldots\}$ of all flows encountered by the SSP algorithm by $\mathcal{F}_0(G)$. Furthermore, we set $\mathcal{F}(G) = \mathcal{F}_0(G) \setminus \{f^0\}$. (We omit the parameter G if it is clear from the context.)

Let us remark that we have not specified in Algorithm 1 which path is chosen if the shortest s-t path is not unique. This is not important for our analysis because we will see in Section 4.2.4 that this happens only with probability 0 in our probabilistic model. We can therefore assume $\mathcal{F}_0(G)$ to be well-defined.

By f^0 and $f^{\max}$, we denote the empty flow and the maximum flow, i.e., the flow that assigns 0 to all edges e and the flow of maximum value encountered by the SSP algorithm, respectively.

Let f^{i-1} and f^i be two consecutive flows encountered by the SSP algorithm and let P^i be the shortest path in the residual network $G_{f^{i-1}}$, i.e., the SSP algorithm augments along P^i to increase flow f^{i-1} to obtain flow f^i. We call P^i the *next path* of f^{i-1} and the *previous path* of f^i. As mentioned in Section 1.3.2, to distinguish between edges in the residual network and edges in the original network, we refer to the edges in the residual network as ‘arcs’, whereas we refer to the edges in the original network as ‘edges’.

For a given arc e in a residual network G_f, we denote by e_0 the corresponding edge in the original network G, i.e., $e_0 = e$ if $e \in E$ (i.e. e is a forward arc) and $e_0 = e^{-1}$ if $e \notin E$ (i.e. e is a backward arc). An arc e is called *empty* (with respect to some residual network G_f) if e belongs to G_f, but e^{-1} does not. Empty arcs e are either forward arcs that do not carry flow or backward arcs whose corresponding edge e_0 carries as much flow as possible. We say that an arc *becomes saturated* (during an augmentation) when it is contained in the current augmenting path, but it does not belong to the residual network that we obtain after this augmentation.

In the remainder, a *path* is always a simple directed path. Let P be a path, and let u and v be contained in P in this order. By $u \overset{P}{\rightsquigarrow} v$, we refer to the sub-path of P starting from node u going to node v, by $\overleftarrow{P}$ we refer to the path we obtain by reversing the direction of each edge of P. We denote by $\overleftrightarrow{G} = (V, E \cup E^{-1})$ for $E^{-1} = \{e^{-1} : e \in E\}$ the flow network that consists of all forward arcs and backward arcs.

4.2.3 Outline of Our Approach

Our analysis of the SSP algorithm is based on the following idea: We identify a flow $f^i \in \mathcal{F}_0$ with a real number by mapping f^i to the length ℓ^i of the previous path P^i of f^i. The flow f^0 is identified with $\ell^0 = 0$. In this way, we obtain a sequence $L = (\ell^0, \ell^1, \ldots)$ of real numbers. We show that this sequence is strictly monotonically increasing with probability 1. Since all costs are drawn from the interval $[0, 1]$, each element of L is from the interval $[0, n]$. To count the number of elements of L, we partition the interval $[0, n]$ into small sub-intervals of length ε and sum up the number of elements of L in these intervals. By linearity of expectation, this approach carries over to the expected number of elements of L. If ε is very small, then – with sufficiently high probability – each interval contains at most one element. If this is the case then it suffices to bound the probability that an element of L falls into some interval $(d, d + \varepsilon]$ because this probability equals the expected number of elements in $(d, d + \varepsilon]$.

To do so, we assume for the moment that there is an integer i such that $\ell^i \in (d, d + \varepsilon]$. By the previous assumption that for any interval of length ε there is at most one path whose length is within this interval, we obtain that $\ell^{i-1} \leq d$. We show that the augmenting path P^i uses an empty arc e. Moreover, we will see that we can reconstruct the flow f^{i-1} and the path P^i without knowing the costs of edge e_0 that corresponds to arc e in the original network. This allows us to use the principle of deferred decisions: to bound the probability that ℓ^i falls into the interval $(d, d + \varepsilon]$, we first reveal all costs $c_{e'}$ with $e' \neq e_0$. Then P^i is known and its length, which equals ℓ^i, can be expressed as a linear function $\kappa + c_{e_0}$ or $\kappa - c_{e_0}$ for a known constant κ. Consequently, the probability that ℓ^i falls into the interval $(d, d + \varepsilon]$ is bounded by $\varepsilon\phi$, as the probability density of c_{e_0} is bounded by ϕ. Since the arc e

is not always the same, we have to apply a union bound over all $2m$ possible arcs. Summing up over all n/ε intervals the expected number of flows encountered by the SSP algorithm can be bounded by roughly $(n/\varepsilon) \cdot 2m \cdot \varepsilon\phi = 2mn\phi$.

There are some parallels to the analysis of the smoothed number of Pareto-optimal solutions in bicriteria linear optimization problems by Beier and Vöcking [8], although we have only one objective function. In this context, we would call f^i the loser, f^{i-1} the winner, and the difference $\ell^i - d$ the loser gap. Beier and Vöcking's analysis is also based on the observation that the winner (which in their analysis is a Pareto-optimal solution and not a flow) can be reconstructed when all except for one random coefficients are revealed. While this reconstruction is simple in the setting of bicriteria optimization problems, the reconstruction of the flow f^{i-1} in our setting is significantly more challenging and a main difficulty in our analysis.

4.2.4 Proof of the Upper Bound

Before we start with the analysis, note that due to our transformation of the general minimum-cost flow problem to a single-source-single-sink minimum-cost flow problem the cost perturbations only affect the original edges. The costs of the auxiliary edges are not perturbed but set to 0. Thus, we will slightly deviate from what we described in the outline by treating empty arcs corresponding to auxiliary edges separately.

The SSP algorithm is in general not completely specified, since at some point during the run of the algorithm there could exist multiple shortest s-t paths in the residual network of the current flow. The SSP algorithm then allows any of them to be chosen as the next augmenting path. Due to Lemma 4.2.3 and Property 4.2.4 we can assume that this is not the case in our setting and that the SSP algorithm is completely specified. In the following we use the concepts of possible residual networks, possible paths, and possible cycles. For their definitions we refer to Section 1.3.

Lemma 4.2.3. *For any real $\varepsilon > 0$ the probability that there are two nodes u and v and two distinct possible u-v paths whose lengths differ by at most ε is bounded from above by $2n^{2n}\varepsilon\phi$.*

Proof. Fix two nodes u and v and two distinct possible u-v paths P_1 and P_2. Then there is an edge e such that one of the paths – without loss of generality path P_1 – contains arc e or e^{-1}, but the other one does not. If we fix all edge costs except the cost of edge e, then the length of P_2 is already determined whereas the length of P_1 depends on the cost c_e. Hence, c_e must fall into a fixed interval of length 2ε in order for the path lengths of P_1 and P_2 to differ by at most ε. The probability for this is bounded by $2\varepsilon\phi$ because c_e is chosen according to a density function that is bounded from above by ϕ. A union bound over all pairs (u, v) and all possible u-v paths concludes the proof. □

The proof also shows that we can assume that there is no s-t path of length 0 and according to Lemma 4.2.3 we can assume that the following property holds since it holds with a probability of 1.

Property 4.2.4. *For any nodes u and v the lengths of all possible u-v paths are pairwise distinct.*

Lemma 4.2.5. *Let $d^i(v)$ denote the distance from s to node v and $\bar{d}^i(v)$ denote the distance from node v to t in the residual network G_{f^i}. Then the sequences $d^0(v), d^1(v), d^2(v), \ldots$ and $\bar{d}^0(v), \bar{d}^1(v), \bar{d}^2(v), \ldots$ are monotonically increasing for every $v \in V$.*

Proof. We only show the proof for the sequence $d^0(v), d^1(v), d^2(v), \ldots$. The proof for the sequence $\bar{d}^0(v), \bar{d}^1(v), \bar{d}^2(v), \ldots$ can be shown analogously. Let $i \geq 0$ be an arbitrary integer. We show $d^i(v) \leq d^{i+1}(v)$ by induction on the depth of node v in the shortest path tree T_{i+1} of the residual network $G_{f^{i+1}}$ rooted at s. For the root s, the claim holds since $d^i(s) = d^{i+1}(s) = 0$. Now assume that the claim holds for all nodes up to a certain depth k, consider a node v with depth $k+1$, and let u denote its parent. Consequently, $d^{i+1}(v) = d^{i+1}(u) + c_e$ for $e = (u, v)$. If arc e has been available in G_{f^i}, then $d^i(v) \leq d^i(u) + c_e$. If not, then the SSP algorithm must have augmented along e^{-1} in step $i{+}1$ to obtain flow f^{i+1} and, hence, $d^i(u) = d^i(v) + c_{e^{-1}} = d^i(v) - c_e$. In both cases the inequality $d^i(v) \leq d^i(u) + c_e$ holds. Applying the induction hypothesis for node u, we obtain $d^i(v) \leq d^i(u) + c_e \leq d^{i+1}(u) + c_e = d^{i+1}(v)$. □

Definition 4.2.6. *For a flow $f^i \in \mathcal{F}_0$, we denote by $\ell_-^G(f^i)$ and $\ell_+^G(f^i)$ the length of the previous path P^i and the next path P^{i+1} of f^i, respectively. By convention, we set $\ell_-^G(f^0) = 0$ and $\ell_+^G(f^{\max}) = \infty$. If the network G is clear from the context, then we simply write $\ell_-(f^i)$ and $\ell_+(f^i)$. By $\mathscr{C}$ we denote the cost function that maps real numbers x from the interval $\big[0, |f^{\max}|\big]$ to the costs of the cheapest flow f with value x, i.e., $\mathscr{C}(x) = \min\{c(f) : |f| = x\}$.*

The lengths $\ell_-(f^i)$ correspond to the lengths ℓ^i mentioned in the outline. The apparent notational overhead is necessary for formal correctness. In Lemma 4.2.8, we will reveal a connection between the values $\ell_-(f^i)$ and the function $\mathscr{C}$. Based on this, we can focus on analyzing the function $\mathscr{C}$.

Lemma 4.2.5 implies in particular that the distance from the source s to the sink t is monotonically increasing, which yields the following corollary.

Corollary 4.2.7. *Let $f^i, f^j \in \mathcal{F}_0$ be two flows with $i < j$. Then $\ell_-(f^i) \leq \ell_-(f^j)$.*

Lemma 4.2.8. *The function $\mathscr{C}$ is continuous, monotonically increasing, and piecewise linear, and the break points of the function are the values of the flows $f \in \mathcal{F}_0$ with $\ell_-(f) < \ell_+(f)$. For each flow $f \in \mathcal{F}_0$, the slopes of $\mathscr{C}$ to the left and to the right of $|f|$ equal $\ell_-(f)$ and $\ell_+(f)$, respectively.*

Proof. The proof follows from Theorem 4.2.1 and the observation that the costs of the flow are linearly increasing when gradually increasing the flow along the shortest path in the residual network until at least one arc becomes saturated. The slope of the cost function is given by the length of that path. □

Example 4.2.9. *Consider the flow network depicted in Figure 4.1. The cost c_e and the capacity u_e of an edge e are given by the notation c_e, u_e. For each step of the SSP algorithm, Figure 4.3 lists the relevant part of the augmenting path (excluding s, s', t', and t), its length, the amount of flow that is sent along that path, and the arcs that become saturated. As can be seen in the table, the values $|f|$ of the encountered flows $f \in \mathcal{F}_0$ are 0, 2, 3, 5, 7, 10, and 12. These are the breakpoints of the cost function $\mathscr{C}$, and the lengths of the augmenting paths equal the slopes of $\mathscr{C}$ (see Figure 4.2).*

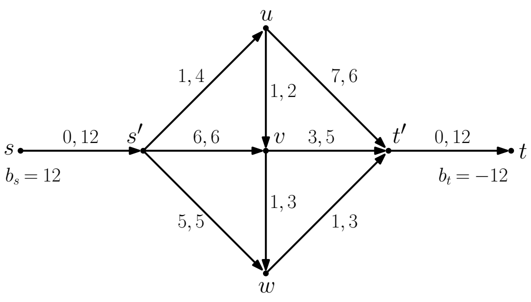


Figure 4.1: Minimum-cost flow network with master source s and master sink t.

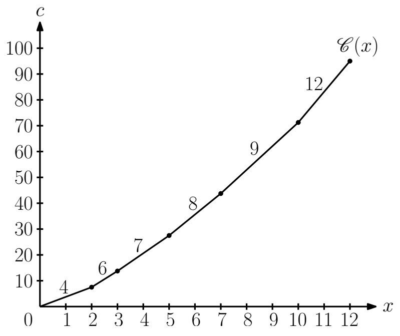


Figure 4.2: Cost function $\mathscr{C}$.

step	1	2	3	4	5	6
path	u, v, w	w	w, v	u	v	v, u
path length	4	6	7	8	9	12
amount of flow	2	1	2	2	3	2
saturated arcs	(u, v)	(w, t')	(w, v)	(s', u)	(v, t')	(v, u)

Figure 4.3: The augmenting paths for Example 4.2.9.

With the following definition, we lay the foundation for distinguishing between original edges with perturbed costs and auxiliary edges whose costs are set to 0.

Definition 4.2.10. *Let $f \in \mathcal{F}_0$ be an arbitrary flow. An empty arc e in the residual network G_f that does not correspond to an auxiliary edge is called a* good arc*. We call f a* good flow *if $f \neq f^0$ and if the previous path of f contains a good arc in the previous residual network. Otherwise, f is called a* bad flow.

Before we can derive a property of good arcs that are contained in the previous path of good flows, we need to show that for each flow value the minimum-cost flow is unique with probability 1.

Lemma 4.2.11. *For any real $\varepsilon > 0$ the probability that there exists a possible cycle whose costs lie in $[0, \varepsilon]$ is bounded from above by $2n^{2n}\varepsilon\phi$.*

Proof. Assume that there exists a cycle K whose costs lie in $[0, \varepsilon]$. Then K contains two nodes u and v and consists of a u-v path P_1 and a v-u path P_2. Then P_1 and $\overleftarrow{P_2}$ are two distinct u-v paths. Since K has costs in $[0, \varepsilon]$, the costs of P_1 and $\overleftarrow{P_2}$ differ by at most ε. Now Lemma 4.2.3 concludes the proof. □

According to Lemma 4.2.11 we can assume that the following property holds since it holds with a probability of 1.

Property 4.2.12. *There exists no possible cycle with costs 0.*

With Property 4.2.12 we can show that the minimum-cost flow is unique for each value.

Lemma 4.2.13. *For each value $B \in \mathbb{R}^+$ there either exists no flow f with $|f| = B$ or there exists a unique minimum-cost flow f with $|f| = B$.*

Proof. Assume that there exists a value $B \in \mathbb{R}^+$ and two distinct minimum-cost flows f and f' with $|f| = |f'| = B$. Let $E_\Delta := \{e \in E \mid f_e \neq f'_e\}$ be the set of edges on which f and f' differ. We show in the following that the set E_Δ contains at least one undirected cycle K. Since f and f' are distinct flows, the set E_Δ cannot be empty. For $v \in V$, let us denote by $f_-(v) = \sum_{e=(u,v)\in E} f_e$ the flow entering v and by $f_+(v) = \sum_{e=(v,w)\in E} f_e$ the flow going out of v ($f'_-(v)$ and $f'_+(v)$ are defined analogously). Flow conservation and $|f| = |f'|$ imply $f_-(v) - f'_-(v) = f_+(v) - f'_+(v)$ for all $v \in V$. Now let us assume E_Δ does not contain an undirected cycle. In this case there must exist a node $v \in V$ with exactly one incident edge in E_Δ. We will show that this cannot happen.

Assume $f_-(v) - f'_-(v) \neq 0$ for some $v \in V$. Then the flows f and f' differ on at least one edge $e = (u, v) \in E$. Since this case implies $f_+(v) - f'_+(v) \neq 0$, they also differ on at least one edge $e' = (v, w) \in E$ and both these edges belong to E_Δ. It remains to consider nodes $v \in V$ with $f_-(v) - f'_-(v) = f_+(v) - f'_+(v) = 0$ and at least one incident edge in E_Δ. For such a node v there exists an edge $e = (u, v) \in E$ (or $e = (v, w) \in E$) with $f_e \neq f'_e$. It follows $\sum_{e'=(u',v)\in E, e'\neq e} f_{e'} - f'_{e'} \neq 0$ (or $\sum_{e'=(v,w')\in E, e'\neq e} f_{e'} - f'_{e'} \neq 0$) which implies that there exists another edge $e' = (u', v) \neq e$ (or $e = (v, w') \neq e$) with $f_{e'} \neq f'_{e'}$.

For the flow $f'' = \frac{1}{2}f + \frac{1}{2}f'$, which has the same costs as f and f' and is hence a minimum-cost flow with $|f''| = B$ as well, we have $f''(e) \in (0, u_e)$ for all $e \in E_\Delta$. The flow f'' can therefore be augmented in both directions along K. Due to Property 4.2.12, augmenting f'' in one of the two directions along K will result in a better flow. This is a contradiction. □

Now we derive a property of good arcs that are contained in the previous path of good flows. This property allows us to bound the probability that one of the lengths $\ell_-(f^i)$ falls into a given interval of length ε.

Lemma 4.2.14. *Let $f \in \mathcal{F}_0$ be a predecessor of a good flow for which $\ell_-^G(f) < \ell_+^G(f)$ holds. Additionally, let e be a good arc in the next path of f, and let e_0 be the edge in G that corresponds to e. Now change the cost of e_0 to $c'_{e_0} = 1$ ($c'_{e_0} = 0$) if $e_0 = e$ ($e_0 = e^{-1}$), i.e., when e is a forward (backward) arc. In any case, the cost of arc e increases. We denote the resulting flow network by G'. Then $f \in \mathcal{F}_0(G')$. Moreover, the inequalities $\ell_-^{G'}(f) \le \ell_-^G(f) < \ell_+^G(f) \le \ell_+^{G'}(f)$ hold.*

Proof. Let $\mathscr{C}$ and $\mathscr{C}'$ be the cost functions of the original network G and the modified network G', respectively. Both functions are of the form described in Lemma 4.2.8. In particular, they are continuous and the breakpoints correspond to the values of the flows $\tilde{f} \in \mathcal{F}_0(G)$ and $\hat{f} \in \mathcal{F}_0(G')$ with $\ell_-^G(\tilde{f}) < \ell_+^G(\tilde{f})$ and $\ell_-^{G'}(\hat{f}) < \ell_+^{G'}(\hat{f})$, respectively.

We start by analyzing the case $e_0 = e$. In this case, we set $\mathscr{C}'' = \mathscr{C}'$ and observe that increasing the cost of edge e_0 to 1 cannot decrease the costs of any flow in G. Hence, $\mathscr{C}'' \ge \mathscr{C}$. Since flow f does not use arc e, its costs remain unchanged, i.e., $\mathscr{C}''(|f|) = \mathscr{C}(|f|)$.

If $e_0 = e^{-1}$, then we set $\mathscr{C}'' = \mathscr{C}' + \Delta_{e_0}$ for $\Delta_{e_0} = u_{e_0} \cdot c_{e_0}$. This function is also piecewise linear and has the same breakpoints and slopes as $\mathscr{C}'$. Since the flow on edge e_0 cannot exceed the capacity u_{e_0} of edge e_0 and since the cost on that edge has been reduced by c_{e_0} in G', the costs of each flow are reduced by at most Δ_{e_0} in G'. Furthermore, this gain is only achieved for flows that entirely use edge e_0 like f does. Hence, $\mathscr{C}'' \ge \mathscr{C}$ and $\mathscr{C}''(|f|) = \mathscr{C}(|f|)$.

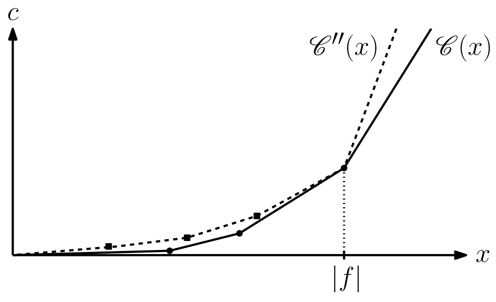


Figure 4.4: Cost function $\mathscr{C}$ and function $\mathscr{C}''$.

Algorithm 2 Reconstruct(e, d).

```
1: let $e_0$ be the edge that corresponds to arc $e$ in the original network $G$
2: change the cost of edge $e_0$ to $c'_{e_0} = 1$ if $e$ is a forward arc or to $c'_{e_0} = 0$ if $e$ is a
   backward arc
3: start running the SSP algorithm on the modified network $G'$
4: stop when the length of the shortest $s$-$t$ path in the residual network of the
   current flow $f'$ exceeds $d$
5: output $f'$
```

Due to $\mathscr{C}'' \geq \mathscr{C}$, $\mathscr{C}''(|f|) = \mathscr{C}(|f|)$, and the form of both functions, the left-hand derivative of $\mathscr{C}''$ at $|f|$ is at most the left-hand derivative of $\mathscr{C}$ at $|f|$ (see Figure 4.4). Since $|f|$ is a breakpoint of $\mathscr{C}$, this implies that $|f|$ is also a breakpoint of $\mathscr{C}''$ and that the slope of $\mathscr{C}''$ to the left of $|f|$ is at most the slope of $\mathscr{C}$ to the left of $|f|$. For the same reasons, the right-hand derivative of $\mathscr{C}''$ at $|f|$ is at least the right-hand derivative of $\mathscr{C}$ at $|f|$ and the slope of $\mathscr{C}''$ to the right of $|f|$ is at least the slope of $\mathscr{C}$ to the right of $|f|$. These properties carry over to $\mathscr{C}'$. Hence, $\mathcal{F}_0(G')$ contains a flow f' with $|f'| = |f|$. Since f is a minimum-cost flow with respect to c, f' is a minimum-cost flow with respect to c', we have $c'(f) = c(f)$ and $c'(f^*) \geq c(f^*)$ for all possible flows f^*, Lemma 4.2.13 yields $f = f'$ and therefore $f \in \mathcal{F}_0(G')$. Recalling the fact that the slopes correspond to shortest s-t path lengths, the stated chain of inequalities follows. □

Lemma 4.2.14 suggests Algorithm 2 (Reconstruct) for reconstructing a flow f based on a good arc e that belongs to the shortest path in the residual network G_f and on a threshold $d \in \big[\ell_-(f), \ell_+(f)\big)$. The crucial fact that we will later exploit is that for this reconstruction the cost c_{e_0} of edge e_0 does not have to be known. (Note that we only need Reconstruct for the analysis in order to show that the flow f can be reconstructed.)

Corollary 4.2.15. *Let $f \in \mathcal{F}_0$ be a predecessor of a good flow, let e be a good arc in the next path of f, and let $d \in \big[\ell_-(f), \ell_+(f)\big)$ be a real number. Then* Reconstruct(e, d) *outputs flow f.*

Proof. By applying Lemma 4.2.14, we obtain $f \in \mathcal{F}_0(G')$ and $\ell_-^{G'}(f) \leq d < \ell_+^{G'}(f)$. Together with Corollary 4.2.7, this implies that Reconstruct(e, d) does not stop before encountering flow f and stops once it encounters f. Hence, Reconstruct(e, d) outputs flow f. □

Corollary 4.2.15 is an essential component of the proof of Theorem 4.2.2 but it only describes how to reconstruct predecessor flows f of good flows with $\ell_-(f) < \ell_+(f)$. In the next part of this section we show that most of the flows are good flows and that, with a probability of 1, the inequality $\ell_-(f) < \ell_+(f)$ holds for any flow $f \in \mathcal{F}_0$.

Lemma 4.2.16. *In any step of the SSP algorithm, any s-t path in the residual network contains at least one empty arc.*

Proof. The claim is true for the empty flow f^0. Now consider a flow $f^i \in \mathcal{F}$, its predecessor flow f^{i-1}, the path P^i, which is a shortest path in the residual network $G_{f^{i-1}}$, and an arbitrary s-t path P in the current residual network G_{f^i}. We show that at least one arc in P is empty.

For this, fix one arc $e = (x, y)$ from P^i that is not contained in the current residual network G_{f^i} since it became saturated by the augmentation along P^i. Let v be the first node of P that occurs in the sub-path $y \overset{P^i}{\rightsquigarrow} t$ of P^i, and let u be the last node in the sub-path $s \overset{P}{\rightsquigarrow} v$ of P that belongs to the sub-path $s \overset{P^i}{\rightsquigarrow} x$ of P^i (see Figure 4.5). By the choice of u and v, all nodes on the sub-path $P' = u \overset{P}{\rightsquigarrow} v$ of P except u and v do not belong to P^i. Hence, the arcs of P' are also available in the residual network $G_{f^{i-1}}$ and have the same capacity in both residual networks $G_{f^{i-1}}$ and G_{f^i}.

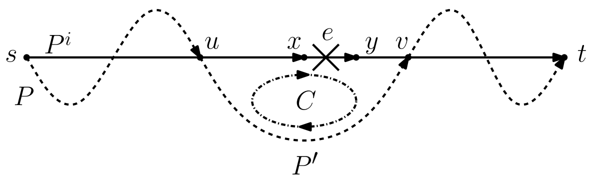


Figure 4.5: Paths P and P^i in the residual network G_{f^i}.

In the remainder of this proof, we show that at least one arc of P' is empty. Assume to the contrary that none of the arcs is empty in G_{f^i} and, hence, in $G_{f^{i-1}}$. This implies that, for each arc $e \in P'$, the residual network $G_{f^{i-1}}$ also contains the arc e^{-1}. Since P^i is the shortest s-t path in $G_{f^{i-1}}$ and since the lengths of all possible s-t paths are pairwise distinct, the path $s \overset{P^i}{\rightsquigarrow} u \overset{P}{\rightsquigarrow} v \overset{P^i}{\rightsquigarrow} t$ is longer than P^i. Consequently, the path $P' = u \overset{P}{\rightsquigarrow} v$ is longer than the path $u \overset{P^i}{\rightsquigarrow} v$. This contradicts the fact that flow f^{i-1} is optimal since the arcs of path $u \overset{P^i}{\rightsquigarrow} v$ combined with the reverse arcs e^{-1} of all the arcs e of path P' form a directed cycle C in $G_{f^{i-1}}$ of negative costs. □

We want to partition the interval $[0, n]$ into small sub-intervals of length ε and treat the number of lengths $\ell\_(f^i)$ that fall into a given sub-interval as a binary random variable. This may be wrong if there are two possible s-t paths whose lengths differ by at most ε. In this case whose probability tends to 0 (see Lemma 4.2.3) we will simply bound the number of augmentation steps of the SSP algorithm by a worst-case bound according to the following lemma.

Lemma 4.2.17. *The number $|\mathcal{F}_0|$ of flows encountered by the SSP algorithm is bounded by 3^{m+n}.*

Proof. We call two possible residual networks equivalent if they contain the same arcs. Equivalent possible residual networks have the same shortest s-t path in common. The length of this path is also the same. Assume that for two distinct flows

$f^i, f^j \in \mathcal{F}_0$ with $i < j$, the residual networks G_{f^i} and G_{f^j} are equivalent. We then have $\ell_-(f^{i+1}) = \ell_+(f^i) = \ell_+(f^j) = \ell_-(f^{j+1})$ and due to Corollary 4.2.7, $\ell_-(f^{i+1}) = \ell_-(f^k) = \ell_-(f^{j+1})$ for all $i < k \leq j+1$. Property 4.2.4 then implies $P^{i+1} = P^k$ for all $i < k \leq j+1$ and especially $P^{i+1} = P^{i+2}$, which is a contradiction. Therefore the number of equivalence classes is bounded by 3^{m+n} since there are m original edges and at most n auxiliary edges. This completes the proof. □

Lemma 4.2.18. *There are at most n bad flows $f \in \mathcal{F}$.*

Proof. According to Lemma 4.2.16, the augmenting path contains an empty arc e in each step. If e is an arc that corresponds to an auxiliary edge (this is the only case when e is not a good arc), then e is not empty after the augmentation. Since the SSP algorithm does not augment along arcs e^{-1} if e is an arc that corresponds to an auxiliary edge, non-empty arcs that correspond to auxiliary edges cannot be empty a second time. Thus, there can be at most n steps where the augmenting path does not contain a good arc. This implies that there are at most n bad flows $f \in \mathcal{F}$. □

We can now bound the probability that there is a flow $f^i \in \mathcal{F}$ whose previous path's length $\ell_-(f^i)$ falls into a given sub-interval of length ε. Though we count bad flows separately, they also play a role in bounding the probability that there is a *good* flow $f^i \in \mathcal{F}$ such that $\ell_-(f^i)$ falls into a given sub-interval of length ε.

Lemma 4.2.19. *For a fixed real $d \geq 0$, let $\mathsf{E}_{d,\varepsilon}$ be the event that there is a flow $f \in \mathcal{F}$ for which $\ell_-(f) \in (d, d+\varepsilon]$, and let $B_{d,\varepsilon}$ be the event that there is a bad flow $f' \in \mathcal{F}$ for which $\ell_-(f') \in (d, d+\varepsilon]$. Then the probability of $\mathsf{E}_{d,\varepsilon}$ can be bounded by $\mathbb{P}(\mathsf{E}_{d,\varepsilon}) \leq 2m\varepsilon\phi + 2 \cdot \mathbb{P}(B_{d,\varepsilon})$.*

Proof. Let $A_{d,\varepsilon}$ be the event that there is a good flow $f \in \mathcal{F}$ for which $\ell_-(f) \in (d, d+\varepsilon]$. Since $\mathsf{E}_{d,\varepsilon} = A_{d,\varepsilon} \cup B_{d,\varepsilon}$, it suffices to show that $\mathbb{P}(A_{d,\varepsilon}) \leq 2m\varepsilon\phi + \mathbb{P}(B_{d,\varepsilon})$. Consider the event that there is a good flow whose previous path's length lies in the interval $(d, d+\varepsilon]$. Among all these good flows, let $\hat{f}$ be the one with the smallest value $\ell_-(\hat{f})$, i.e., $\hat{f}$ is the first good flow f encountered by the SSP algorithm for which $\ell_-(f) \in (d, d+\varepsilon]$, and let f^* be its previous flow. Flow f^* always exists since $\hat{f}$ cannot be the empty flow f^0. Corollary 4.2.7 and Property 4.2.4 yield $\ell_-(f^*) < \ell_-(\hat{f})$. Thus, there can only be two cases: If $\ell_-(f^*) \in (d, d+\varepsilon]$, then f^* is a bad flow by the choice of $\hat{f}$ and, hence, event $B_{d,\varepsilon}$ occurs. The interesting case, which we consider now, is when $\ell_-(f^*) \leq d$ holds. If this is true, then $d \in [\ell_-(f^*), \ell_+(f^*))$ due to $\ell_+(f^*) = \ell_-(\hat{f})$.

As $\hat{f}$ is a good flow, the shortest path in the residual network G_{f^*} contains a good arc $e = (u, v)$. Applying Corollary 4.2.15 we obtain that we can reconstruct flow f^* by calling $\mathsf{Reconstruct}(e, d)$. The shortest s-t path P in the residual network G_{f^*} is the previous path of $\hat{f}$ and its length equals $\ell_-(\hat{f})$. Furthermore, P is of the form $s \overset{P}{\rightsquigarrow} u \to v \overset{P}{\rightsquigarrow} t$, where $s \overset{P}{\rightsquigarrow} u$ and $v \overset{P}{\rightsquigarrow} t$ are shortest paths in G_{f^*} from s to u and from v to t, respectively. These observations yield

$$A_{d,\varepsilon} \subseteq \bigcup_{e \in E} R_{e,d,\varepsilon} \cup \bigcup_{e \in E} R_{e^{-1},d,\varepsilon} \cup B_{d,\varepsilon} \,,$$

where $R_{e,d,\varepsilon}$ for some arc $e = (u, v)$ denotes the following event: The event $R_{e,d,\varepsilon}$ occurs if $\ell \in (d, d + \varepsilon]$, where ℓ is the length of the shortest s-t path that uses arc e in G_f, the residual network of the flow f obtained by calling the procedure $\mathsf{Reconstruct}(e, d)$. Therefore, the probability of event $A_{d,\varepsilon}$ is bounded by

$$\sum_{e \in E} \mathbb{P}(R_{e,d,\varepsilon}) + \sum_{e \in E} \mathbb{P}(R_{e^{-1},d,\varepsilon}) + \mathbb{P}(B_{d,\varepsilon})\,.$$

We conclude the proof by showing $\mathbb{P}(R_{e,d,\varepsilon}) \leq \varepsilon\phi$. For this, let e_0 be the edge corresponding to arc $e = (u, v)$ in the original network. If we fix all edge costs except cost c_{e_0} of edge e_0, then the output f of $\mathsf{Reconstruct}(e, d)$ is already determined. The same holds for the shortest s-t path in G_f that uses arc e since it is of the form $s \rightsquigarrow u \to v \rightsquigarrow t$ where $P_1 = s \rightsquigarrow u$ is a shortest s-u path in G_f that does not use v and where $P_2 = v \rightsquigarrow t$ is a shortest v-t path in G_f that does not use u. The length ℓ of this path, however, depends linearly on the cost c_{e_0}. To be more precise, $\ell = \ell' + c_e = \ell' + \mathrm{sgn}(e) \cdot c_{e_0}$, where ℓ' is the length of P_1 plus the length of P_2 and where

$$\mathrm{sgn}(e) = \begin{cases} +1 & \text{if } e_0 = e\,, \\ -1 & \text{if } e_0 = e^{-1}\,. \end{cases}$$

Hence, ℓ falls into the interval $(d, d+\varepsilon]$ if and only if c_{e_0} falls into some fixed interval of length ε. The probability for this is bounded by $\varepsilon\phi$ as c_{e_0} is drawn according to a distribution whose density is bounded by ϕ. □

Corollary 4.2.20. *The expected number of augmentation steps the SSP algorithm performs is bounded by $2mn\phi + 2n$.*

Proof. Let $X = |\mathcal{F}|$ be the number of augmentation steps of the SSP algorithm. For reals $d, \varepsilon > 0$, let $\mathsf{E}_{d,\varepsilon}$ and $B_{d,\varepsilon}$ be the events defined in Lemma 4.2.19, let $X_{d,\varepsilon}$ be the number of flows $f \in \mathcal{F}$ for which $\ell\_(f) \in (d, d + \varepsilon]$, and let $Z_{d,\varepsilon} = \min\{X_{d,\varepsilon}, 1\}$ be the indicator variable of event $\mathsf{E}_{d,\varepsilon}$.

Since all costs are drawn from the interval $[0, 1]$, the length of any possible s-t path is bounded by n. Furthermore, according to Corollary 4.2.7, all lengths are non-negative (and positive with a probability of 1). Let F_ε denote the event that there are two possible s-t paths whose lengths differ by at most ε. Then, for any positive integer k, we obtain

$$X = \sum_{i=0}^{k-1} X_{i\cdot\frac{n}{k},\frac{n}{k}} \begin{cases} = \sum\limits_{i=0}^{k-1} Z_{i\cdot\frac{n}{k},\frac{n}{k}} & \text{if } F_{\frac{n}{k}} \text{ does not occur}\,, \\ \leq 3^{m+n} & \text{if } F_{\frac{n}{k}} \text{ occurs}\,. \end{cases}$$

Consequently,

$$\begin{aligned}\mathbb{E}(X) &\leq \sum_{i=0}^{k-1} \mathbb{E}(Z_{i\cdot\frac{n}{k},\frac{n}{k}}) + 3^{m+n}\cdot\mathbb{P}(F_{\frac{n}{k}})\\ &= \sum_{i=0}^{k-1} \mathbb{P}(\mathsf{E}_{i\cdot\frac{n}{k},\frac{n}{k}}) + 3^{m+n}\cdot\mathbb{P}(F_{\frac{n}{k}})\\ &\leq 2mn\phi + 2\cdot\sum_{i=0}^{k-1} \mathbb{P}(B_{i\cdot\frac{n}{k},\frac{n}{k}}) + 3^{m+n}\cdot\mathbb{P}(F_{\frac{n}{k}})\\ &\leq 2mn\phi + 2n + 3^{m+n}\cdot\mathbb{P}(F_{\frac{n}{k}})\,.\end{aligned}$$

The second inequality is due to Lemma 4.2.19 whereas the third inequality stems from Lemma 4.2.18. The claim follows since $\mathbb{P}(F_{\frac{n}{k}}) \to 0$ for $k \to \infty$ in accordance with Lemma 4.2.3. □

Now we are almost done with the proof of our main theorem (Theorem 4.2.2).

Proof of Theorem 4.2.2. Since each step of the SSP algorithm runs in time $O(m + n\log n)$ using Dijkstra's algorithm (see, e.g., Korte and Vygen [36] for details), applying Corollary 4.2.20 yields the desired result. □

4.2.5 Smoothed Analysis of the Simplex Method

In this section we describe a surprising connection between our result about the SSP algorithm and the smoothed analysis of the simplex method for linear programming. Spielman and Teng's original smoothed analysis [56] as well as Vershynin's [61] improved analysis are based on the shadow vertex method. To describe this pivot rule, let us consider a linear program with an objective function $z^T x$ and a set of constraints $Ax \leq b$. Let us assume that a non-optimal initial vertex x_0 of the polytope P of feasible solutions is given. The shadow vertex method computes an objective function $u^T x$ that is optimized by x_0. Then it projects the polytope P onto the 2-dimensional plane that is spanned by the vectors z and u. If we assume for the sake of simplicity that P is bounded, then the resulting projection is a polygon Q.

The crucial properties of the polygon Q are as follows: both the projection of x_0 and the projection of the optimal solution $x^\star$ are vertices of Q, and every edge of Q corresponds to an edge of P. The shadow vertex method follows the edges of Q from the projection of x_0 to the projection of $x^\star$. The aforementioned properties guarantee that this corresponds to a feasible walk on the polytope P.

To relate the shadow vertex method and the SSP algorithm, we consider the canonical linear program for the maximum-flow problem with one source and one sink. In this linear program, there is a variable for each edge corresponding to the flow on that edge. The objective function, which is to be maximized, adds the flow on all outgoing edges of the source and subtracts the flow on all incoming edges of the source. There are constraints for each edge ensuring that the flow is non-negative and not larger than the capacity, and there is a constraint for each node except the source and the sink ensuring Kirchhoff's law.

The empty flow x_0 is a vertex of the polytope of feasible solutions. In particular, it is a feasible solution with minimum costs. Hence, letting u be the vector of edge costs is a valid choice in the shadow vertex method. For this choice every feasible flow f is projected to the pair $(|f|, c(f))$. Theorem 4.2.1 guarantees that the cost function depicted in Figure 4.2 forms the lower envelope of the polygon that results from projecting the set of feasible flows. There are two possibilities for the shadow vertex method for the first step: it can choose to follow either the upper or the lower envelope of this polygon. If it decides for the lower envelope, then it will encounter exactly the same sequence of flows as the SSP algorithm.

This means that Theorem 4.2.2 can also be interpreted as a statement about the shadow vertex method applied to the maximum-flow linear program. It says that for this particular class of linear programs, the shadow vertex method has expected polynomial running-time even if the linear program is chosen by an adversary. It suffices to perturb the costs, which determine the projection used in the shadow vertex method. Hence, if the projection is chosen at random, the shadow vertex method is a randomized simplex method with polynomial expected running-time for any flow linear program.

In general, we believe that it is an interesting question to study whether the strong assumption in Spielman and Teng's [56] and Vershynin's [61] smoothed analyses that all coefficients in the constraints are perturbed is necessary. In particular, we find it an interesting open question to characterize for which class of linear programs it suffices to perturb only the coefficients in the objective function or just the projection in the shadow vertex method to obtain polynomial smoothed running-time.

Brunsch and Röglin have studied a related question [14]. They have proved that the shadow vertex method can be used to find short paths between given vertices of a polyhedron. Here, short means that the path length is $O(\frac{mn^2}{\delta^2})$, where n denotes the number of variables, m denotes the number of constraints, and δ is a parameter that measures the flatness of the vertices of the polyhedron. This result is proved by a significant extension of the analysis presented here. Dadush and Hähnle [20] have in turn improved and generalized the results by Brunsch and Röglin.

4.3 Minimum-Mean Cycle Canceling Algorithm

4.3.1 Introduction

In this section we show an upper bound on the smoothed running-time of the minimum-mean cycle canceling (MMCC) algorithm. The MMCC algorithm is based on the well-known optimality criterion that a flow f is optimal if and only if there is no cycle in the corresponding residual network G_f with negative total weight. The MMCC algorithm first computes any feasible starting flow, and then repeatedly augments flow along cycles of minimum-mean cost, until no negative-cost cycles remain. The choice to augment along minimum-mean cost cycles is made, since this guarantees strongly polynomial running-time. When, for example, flow is instead augmented along the cycle of minimum *total* costs, the running-time of the algorithm is not guaranteed to be polynomial. Before we proof our upper bound on the smoothed running-time of the MMCC algorithm, we first give a definition of the algorithm. For a more elaborate introduction to the MMCC algorithm we refer to Ahuja et al. [1].

Definition of the Algorithm

Before we define the MMCC algorithm, we first introduce some notation. For an introduction to the MCF problem and the residual network we refer to Section 1.3. Consider a flow network $G = (V, E)$. Let C be a cycle in a residual network G_f for some flow f. We define the mean costs $\mu(C)$ of cycle C as

$$\mu(C) = \left(\sum_{e \in C} c_e\right) / |C|.$$

Also, we define the minimum-mean cost cycle for flow f as the cycle $C^\star$ in the residual network G_f that has minimum-mean cost. That is,

$$C^\star = \text{argmin}\{\mu(C) \mid C \in \mathcal{C}\},$$

where $\mathcal{C}$ is the set of all cycles in residual network G_f. Finally, we denote the mean costs of the cycle of minimum-mean cost in the residual network G_f by $\mu(f) = \mu(C^\star)$.

We are now ready to define the MMCC algorithm (Algorithm 3).

Algorithm 3 MMCC algorithm.

```
1: **if** a feasible flow exists **then**
2:     find a feasible flow $f^0$ using any maximum-flow algorithm
3: **else**
4:     output that no feasible flow exists
5: **end if**
6: **for** $i = 1, 2, \ldots$ **do**
7:     **if** $G_{f^{i-1}}$ does not contain a negative cycle **then**
8:         output $f^{i-1}$
9:     **end if**
10:    Find a cycle $C$ in $G_{f^{i-1}}$ of minimum-mean cost and maximally* augment flow
       along $C$ to obtain $f^i$
11: **end for**
```

* Since the flow f^i must obey all capacity constraints, the flow is increased by the minimum of $\min\{u_e - f_e^{i-1} \mid e \in C \cap E\}$ and $\min\{f_{e^{-1}}^{i-1} \mid e \in C \text{ and } e^{-1} \in E\}$.

Known Bounds on the Number of Iterations

Goldberg and Tarjan [29] proved in 1989 that the MMCC algorithm runs in strongly polynomial time. Five years later Radzik and Goldberg [46] slightly improved this bound on the running-time and showed that it is tight. In the following we will focus on the number of iterations the MMCC algorithm needs, that is, the number of cycles that have to be canceled. A bound on the number of iterations can easily be extended to a bound on the running-time, by noting that a minimum-mean cycle can be found in $O(nm)$ time, as shown by Karp [33]. The tight bound on the number of iterations that the MMCC algorithm requires is as follows.

Theorem 4.3.1 (Radzik and Goldberg). *The number of iterations required by the MMCC algorithm is bounded by $O(nm^2)$ and this bound is tight.*

To prove our smoothed upper bound on the running-time of the MMCC algorithm we use another result, which states that the absolute value of the mean costs of the cycle that is canceled by the MMCC algorithm, $|\mu(f)|$, decreases by at least a factor 1/2 every mn iterations.

Theorem 4.3.2 (Korte and Vygen [36, Corollary 9.9]). *Every mn iterations of the MMCC algorithm, $|\mu(f)|$ decreases by at least a factor 1/2.*

Our Results

In Section 4.3.2 we prove the following upper bound on the expected number of iterations that the MMCC algorithm requires in the setting of smoothed analysis.

Theorem 4.3.3. *The expected number of iterations that the MMCC algorithm requires is $O\big(mn(n\log(n)+\log(\phi))\big)$.*

For dense graphs, this is an improvement over the $\Theta(m^2n)$ iterations that the MMCC algorithm needs in the worst case, if we consider ϕ a constant (which is reasonable if it models, for example, numerical imprecision or measurement errors). In Section 5.2 we show lower bounds for the smoothed number of iterations that the MMCC algorithm requires.

4.3.2 Proof of the Upper Bound

In this section we prove Theorem 4.3.3. In the proofs in this section we use the concepts of possible residual networks and possible cycles. For their definitions we refer to Section 1.3. As we remarked at the start of this chapter, all logarithms in the following are to the base 2. Let G_{f^0} be the starting initial residual network for some feasible starting flow f^0 for flow network $G = (V, E)$. Like in Section 4.2, we bound the probability that a possible cycle exists with costs very close to 0.

Lemma 4.3.4. *For any real $\varepsilon > 0$ the probability that there exists a possible cycle whose costs lie in $[-\varepsilon, 0]$ is bounded from above by $2n^{2n}\varepsilon\phi$.*

Proof. The proof is analogous to the proof of Lemma 4.2.11. □

The bound on the probability that a possible cycle has negative total costs close to 0 allows us to bound the probability that a possible cycle has negative mean costs close to 0.

Lemma 4.3.5. *For any real $\varepsilon > 0$ the probability that there exists a possible cycle whose mean costs lie in $[-\varepsilon, 0]$ is bounded from above by $2n^{2n+1}\varepsilon\phi$.*

Proof. Follows directly from Lemma 4.3.4 and the fact that the mean costs of a possible cycle C can lie in the interval $[-\varepsilon, 0]$ only if the total costs of C lie in $[-n\varepsilon, 0]$. □

We now show that we can bound the number of iterations that the MMCC algorithm requires, if no possible cycle has negative costs close to 0.

Lemma 4.3.6. *If no possible residual network G_f contains a cycle C with $\mu(C) \in [-\varepsilon, 0[$, then the MMCC algorithm requires at most $mn\lceil\log(1/\varepsilon)\rceil$ iterations.*

Proof. Assume to the contrary that no possible residual network contains a cycle C with $\mu(C) \in [-\varepsilon, 0[$, but the MMCC algorithm needs more than $mn\lceil\log(1/\varepsilon)\rceil$ iterations. Let f^0 denote the starting flow found using any maximum-flow algorithm. Since all edge costs are drawn from the interval $[0, 1]$, we have that $|\mu(f^0)| \leq 1$. According to Theorem 4.3.2, after $i = mn\lceil\log(1/\varepsilon)\rceil$ iterations, it holds for the current flow f^i that $|\mu(f^i)| \leq \varepsilon$. Now either $\mu(f^i) \geq 0$ contradicting the assumption that the MMCC algorithm requires more than $mn\lceil\log(1/\varepsilon)\rceil$ iterations, or $\mu(f^i) \in [-\varepsilon, 0[$ contradicting the assumption that no possible residual network G_f contains a cycle C with $\mu(C) \in [-\varepsilon, 0[$. □

To complete the proof of Theorem 4.3.3 we now use Lemma 4.3.6 to bound the expected number of iterations that the MMCC algorithm requires in the smoothed setting.

Proof of Theorem 4.3.3. Let T be the number of iterations that the MMCC algorithm requires and, for compactness, let $\beta = mn\big((2n+1)\lceil\log(n)\rceil + \lceil\log(\phi)\rceil + 1\big)$. We have

$$\begin{aligned}
\mathbb{E}(T) &= \sum_{t=1}^{\infty} \mathbb{P}(T \geq t) \\
&\leq \sum_{t=1}^{\infty} \mathbb{P}(\text{some } G_f \text{ contains a cycle } C \text{ with } \mu(C) \in [-2^{-\lfloor (t-1)/mn \rfloor}, 0[\,) && (4.1) \\
&\leq \beta + \sum_{t=\beta+1}^{\infty} 2n^{2n+1}\phi 2^{-\lfloor (t-1)/mn \rfloor} && (4.2) \\
&\leq \beta + \sum_{t=0}^{\infty} 2^{-\lfloor t/mn \rfloor} = \beta + mn\sum_{t=0}^{\infty} 2^{-t} = O\big(mn(n\log(n) + \log(\phi))\big). && (4.3)
\end{aligned}$$

Here Equation (4.1) follows from Lemma 4.3.6. Equation (4.2) follows by bounding the probability for the first β terms of the summation by 1 and the probability for the other terms using Lemma (4.3.5). Finally, the first inequality in Equation (4.3) follows from the equality $\log(2n^{2n+1}\phi) = (2n+1)\log(n) + \log(\phi) + 1$.

□

CHAPTER 5

Smoothed Lower Bounds for Minimum-Cost Flow Algorithms

In this chapter we show lower bounds for the running-time of the successive shortest path (SSP) algorithm, the minimum-mean cycle canceling (MMCC) algorithm, and the network simplex (NS) algorithm in the setting of smoothed analysis. These lower bounds complement the upper bounds for the smoothed running-time of the SSP algorithm and the MMCC algorithm that we have proved in Chapter 4. We show these lower bounds by constructing for each algorithm an instance for which the algorithm requires a running-time equal to the lower bound. The smoothed lower bound for the SSP algorithm (Section 5.1) is by Clemens Rösner from the University of Bonn, and appeared before in his MSc thesis [49]. We include it here for the sake of completeness. An introduction to the model that we use for the smoothed instances of the minimum-cost flow problem can be found in Section 4.1. Introductions to the SSP algorithm and the MMCC algorithm can be found in Sections 4.2.1 and 4.3.1, respectively. We introduce the NS algorithm in Section 5.3.1. We conclude this chapter (Section 5.4) with a discussion of all the upper and lower bounds that we showed for the smoothed running-time of the SSP algorithm, the MMCC algorithm, and the NS algorithm. Like in the previous chapter, all logarithms in this chapter are to the base 2, unless stated otherwise.

5.1 Successive Shortest Path Algorithm

5.1.1 Smoothed Lower Bound

In this section we provide a lower bound on the number of augmentation steps that the successive shortest path (SSP) algorithm requires in the setting of smoothed analysis. For an introduction to the SSP algorithm we refer to Section 4.2.1. We prove the following result.

Theorem 5.1.1. *For given positive integers n, $m \in \{n, \dots, n^2\}$, and $\phi \leq 2^n$ there exists a minimum-cost flow network with $O(n)$ nodes, $O(m)$ edges, and random edge costs with smoothing parameter ϕ for which the SSP algorithm requires $\Omega(m \cdot \min\{n, \phi\} \cdot \phi)$ augmentation steps with probability 1.*

The lower bound from Theorem 5.1.1 and the upper bound from Section 4.2 for the smoothed number of augmentation steps required by the SSP algorithm differ only a small factor. In case $\phi = \Omega(n)$, the lower bound is even tight.

5.1.2 Proof of the Lower Bound

This section is devoted to the proof of Theorem 5.1.1. For given positive integers n, $m \in \{n, \dots, n^2\}$, and $\phi \leq 2^n$ let $k = \lfloor \log \phi \rfloor - 5 = O(n)$ and $M = \min\left\{n, 2^{\lfloor \log \phi \rfloor}/4 - 2\right\} = \Theta(\min\{n, \phi\})$. In the following we assume that $\phi \geq 64$, such that we have $k, M \geq 1$. If $\phi < 64$, the lower bound on the number of augmentation steps from Theorem 5.1.1 reduces to $\Omega(m)$ and a simple flow network like the network G_1 (as explained below, which we will use as initial network in case $\phi \geq 64$) with $O(n)$ nodes, $O(m)$ edges, and uniform edge costs proves the lower bound.

We construct a flow network with $2n + 2k + 2 + 4M = O(n)$ nodes, $m + 2n + 4k - 4 + 8M = O(m)$ edges, and smoothing parameter ϕ for which the SSP algorithm requires $m \cdot 2^{k-1} \cdot 2M = \Theta(m \cdot \phi \cdot \min\{n, \phi\})$ augmentation steps in expectation. To be exact, we show that for any realization of the edge costs for which there do not exist multiple paths with exactly the same costs (Property 4.2.4) the SSP algorithm requires that many iterations. Since this happens with probability 1, we will assume in the following that Property 4.2.4 holds without further mention.

For the sake of simplicity we consider edge cost densities $g_e \colon [0, \phi] \to [0, 1]$ instead of $g_e \colon [0, 1] \to [0, \phi]$. This model is equivalent to the smoothed input model introduced in Section 4.1, because both types of densities can be transformed into each other by scaling by a factor of ϕ and because the behavior of the SSP algorithm is invariant under scaling of the edge costs. Furthermore, our densities g_e will be uniform distributions on intervals I_e with lengths of at least 1. In the remainder of this section we only construct these intervals I_e. Also, all minimum-cost flow networks constructed in this section have a unique source node s and a unique sink node t, which is always clear from the context. The node budgets are defined as $b_v = 0$ for all nodes $v \notin \{s, t\}$ and $-b_t = b_s = \sum_{e=(s,v)} u_e = \sum_{e=(w,t)} u_e$, that is, each b-flow equals a maximum s-t-flow.

The construction of the desired minimum-cost flow network G consists of three steps, which we sketch below and describe in more detail thereafter. Given Property 4.2.4, our choice of distributions for the edge costs ensures that the behavior of the SSP algorithm is the same for every realization of the edge costs. In the following we say that the SSP algorithm *encounters* a path P on a flow network G' if it augments along P when run on G'.

1. In the first step we define a simple flow network G_1 with a source s_1 and a sink t_1 for which the SSP algorithm requires m augmentation steps.

2. In the second step we take a flow network G_i, starting with $i = 1$, as the basis for constructing a larger flow network G_{i+1}. We obtain the new flow network by adding a new source s_{i+1}, a new sink t_{i+1}, and four edges connecting the new source and sink with the old source and sink. Additionally, the latter two

nodes are downgraded to "normal" nodes (nodes with a budget of 0) in G_{i+1} (see Figure 5.2). By a careful choice of the new capacities and cost intervals we can ensure the following property: First, the SSP algorithm subsequently augments along all paths of the form

$$s_{i+1} \to s_i \overset{P}{\rightsquigarrow} t_i \to t_{i+1} \,,$$

where P is an s_i-t_i path encountered by the SSP algorithm when run on the network G_i. Then, it augments along all paths of the form

$$s_{i+1} \to t_i \overset{\overleftarrow{P}}{\rightsquigarrow} s_i \to t_{i+1} \,,$$

where P is again an s_i-t_i path encountered by the SSP algorithm when run on the network G_i. Hence, by adding two nodes and four edges we double the number of iterations that the SSP algorithm requires. For this construction to work we have to double the maximum edge cost of our flow network. Hence, this construction can be repeated $k-1 \approx \log\phi$ times, yielding an additional factor of $2^{k-1} \approx \phi$ for the number of iterations required by the SSP algorithm.

3. In the third step we add a global source s and a global sink t to the flow network G_k constructed in the second step, and add four directed paths of length $M \approx \min\{n, \phi\}$, where each contains M new nodes and has exactly one node in common with G_k. The first path will end in s_k, the second path will end in t_k, the third path will start in s_k, and the fourth path will start in t_k. We also add an arc from s to every new node in the first two paths and an arc from every new node in the last two paths to t (see Figure 5.3). We call the resulting flow network G. By the right choice of the edge costs and capacities we ensure that for each s_k-t_k path P in G_k encountered by the SSP algorithm on G_k the SSP algorithm on G encounters M augmenting paths having P as a sub-path and M augmenting paths having $\overleftarrow{P}$ as a sub-path. In this way, we gain an additional factor of $2M$ for the number of iterations of the SSP algorithm.

Construction of G_1. For the first step, consider two sets $U = \{u_1, \ldots, u_n\}$ and $W = \{w_1, \ldots, w_n\}$ of n nodes and an arbitrary set $E_{UW} \subseteq U \times W$ containing exactly $|E_{UW}| = m$ edges. The initial flow network G_1 is defined as $G_1 = (V_1, E_1)$ for $V_1 = U \cup W \cup \{s_1, t_1\}$ and

$$E_1 = (\{s_1\} \times U) \cup E_{UW} \cup (W \times \{t_1\}) \,.$$

The edges e from E_{UW} have capacity 1 and costs from the interval $I_e = [7, 9]$. The edges $(s_1, u_i), u_i \in U$ have a capacity equal to the out-degree of u_i, the edges $(w_j, t_1), w_j \in W$ have a capacity equal to the in-degree of w_j and both have costs from the interval $I_e = [0, 1]$ (see Figure 5.1). (Remember that we use uniform distributions on the intervals I_e.)

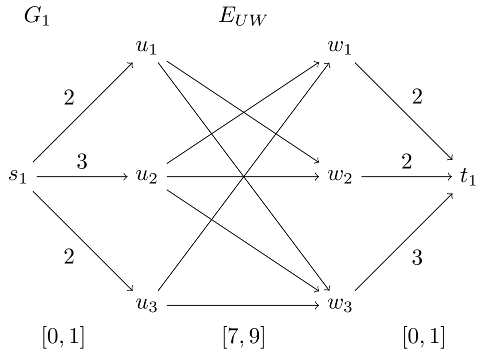


Figure 5.1: Example for G_1 with $n = 3$ and $m = 7$ with capacities different from 1 shown next to the edges and the cost intervals shown below each edge set.

Lemma 5.1.2. *The SSP algorithm requires exactly m iterations on G_1 to obtain a maximum s_1-t_1-flow. Furthermore all augmenting paths it encounters have costs from the interval* $[7, 11]$.

Proof. First we observe that the SSP algorithm augments only along paths that are of the form $s_1 \to u_i \to w_j \to t_1$ for some $u_i \in U$ and $w_j \in W$: Consider an arbitrary augmenting path P the SSP algorithm encounters and assume for contradiction that P is not of this form. Due to the structure of G_1, the first two arcs of P are of the form (s_1, u_i) and (u_i, w_j) for some $u_i \in U$ and $w_j \in W$. The choice of the capacities ensures that the arc (w_j, t_1) cannot be fully saturated if the arc (u_i, w_j) is not. Hence, when the SSP algorithm augments along P, the arc (w_j, t_1) is available in the residual network. Since this arc is not used by the SSP algorithm, the subpath $w_j \overset{P}{\rightsquigarrow} t_1$ has smaller costs than the arc (w_j, t_1). This means that the distance of w_j to the sink t_1 in the current residual network is smaller than in the initial residual network for the zero flow. This contradicts Lemma 4.2.5.

Since every path that the SSP algorithm encounters on G_1 is of the form $s_1 \to u_i \to w_j \to t_1$, every such path consists of two arcs with costs from the interval $[0, 1]$ and one arc with costs from the interval $[7, 9]$. This implies that the total costs of any such path lie in the interval $[7, 11]$.

The choice of capacities ensures that on every augmenting path of the form $s_1 \to u_i \to w_j \to t_1$ the arc (u_i, w_j) is a bottleneck and becomes saturated by the augmentation. As flow is never removed from this arc again, there is a one-to-one correspondence between the paths that the SSP algorithm encounters on G_1 and the edges from E_{UW}. This implies that the SSP algorithm encounters exactly m paths on G_1. □

Construction of G_{i+1} from G_i. Now we describe the second step of our construction more formally. Given a flow network $G_i = (V_i, E_i)$ with a source s_i and a sink t_i, we define $G_{i+1} = (V_{i+1}, E_{i+1})$, where $V_{i+1} = V_i \cup \{s_{i+1}, t_{i+1}\}$ and

$$E_{i+1} = E_i \cup (\{s_{i+1}\} \times \{s_i, t_i\}) \cup (\{s_i, t_i\} \times \{t_{i+1}\}).$$

Let $N_i = 2^{i-1} \cdot m$, which is the value of the maximum s_i-t_i flow in G_i. The new edges $e \in \{(s_{i+1}, s_i), (t_i, t_{i+1})\}$ have capacity $u_e = N_i$ and costs from the interval $I_e = [0, 1]$. The new edges $e \in \{(s_{i+1}, t_i), (s_i, t_{i+1})\}$ also have capacity $u_e = N_i$, but costs from the interval $I_e = [2^{i+3} - 1, 2^{i+3} + 1]$ (see Figure 5.2).

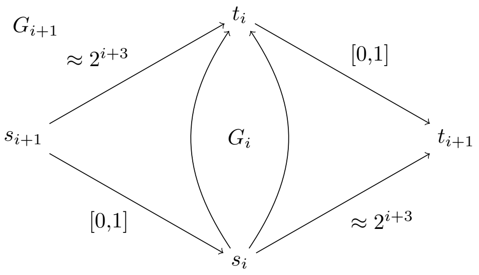


Figure 5.2: G_{i+1} with G_i as sub-graph with edge costs next to the edges.

Next we analyze how many iterations the SSP algorithm requires to reach a maximum s_{i+1}-t_{i+1} flow when run on the network G_{i+1}. Before we can start with this analysis, we prove the following property of the SSP algorithm.

Lemma 5.1.3. *After augmenting flow via a cheapest v-w-path P in a network without a cycle with negative total costs, $\overleftarrow{P}$ is a cheapest w-v-path.*

Proof. Since we augmented along P, all arcs of $\overleftarrow{P}$ will be part of the residual network. $\overleftarrow{P}$ will therefore be a feasible w-v-path. Assume that after augmenting along P there exists a w-v-path P' that is cheaper than $\overleftarrow{P}$. Let us take a look at the multi-set $X = P \cup P'$, which contains every arc $e \in P \cap P'$ twice. The total costs of this multi-set are negative because

$$c(P) + c(P') = -c(\overleftarrow{P}) + c(P') < 0$$

by the assumption that P' is cheaper than $\overleftarrow{P}$. Furthermore, for each node the number of incoming and outgoing arcs from X is the same. This property is preserved if we delete all pairs of a forward arc e and the corresponding backward arc e^{-1} from X, resulting in a multi-set $X' \subseteq X$. The total costs of the arcs in X' are negative because they equal the total costs of the arcs in X.

For every arc $e \in X$ that did not have positive residual capacity before augmenting along P, the arc e^{-1} must be part of P and therefore be part of X as well. This is due to the fact that only for arcs e with $e^{-1} \in P$ the residual capacity increases when augmenting along P. Since all such pairs of arcs are deleted, the set X' will only contain arcs that had a positive residual capacity before augmenting along P. Since each node has the same number of outgoing and incoming arcs from X', we can partition X' into subsets, where the arcs in each subset form a cycle. Since the total costs of all arcs are negative at least one of these cycles has to have negative costs, which is a contradiction. □

Since during the execution of the SSP algorithm all residual networks do not contain negative cycles, Lemma 5.1.3 always applies.

Lemma 5.1.4. *Let $i \geq 1$. All s_i-t_i-paths that the SSP algorithm encounters when run on the network G_i have costs from the interval $[7, 2^{i+3}-5]$. Furthermore the SSP algorithm encounters twice as many paths on the network G_{i+1} as on the network G_i.*

Proof. We prove the first half of the lemma by induction over i. In accordance with Lemma 5.1.2, all paths the SSP algorithm encounters on G_1 have costs from the interval $[7, 11] = [7, 2^4 - 5]$.

Now assume that all paths that the SSP algorithm encounters on G_i, for some $i \geq 1$, have costs from the interval $[7, 2^{i+3} - 5]$. We distinguish between three different kinds of s_{i+1}-t_{i+1}-paths in G_{i+1}.

Definition 5.1.5. *We classify the possible s_{i+1}-t_{i+1}-paths P in G_{i+1} as follows.*

1. *If $P = s_{i+1} \rightarrow s_i \rightsquigarrow t_i \rightarrow t_{i+1}$, then P is called a* type-1-path*.*
2. *If $P = s_{i+1} \rightarrow s_i \rightarrow t_{i+1}$ or $P = s_{i+1} \rightarrow t_i \rightarrow t_{i+1}$, then P is called a* type-2-path*.*
3. *If $P = s_{i+1} \rightarrow t_i \rightsquigarrow s_i \rightarrow t_{i+1}$, then P is called a* type-3-path*.*

For any type-2-path P we have

$$c(P) \in [0 + (2^{i+3} - 1), 1 + (2^{i+3} + 1)] = [2^{i+3} - 1, 2^{i+3} + 2] \subseteq [7, 2^{i+4} - 5]\,.$$

Since due to Lemma 4.2.5 the distance from t_i to t_{i+1} does not decrease during the run of the SSP algorithm, the SSP algorithm will only augment along a type-3-path P once the arc (t_i, t_{i+1}) is saturated. Otherwise, the t_i-t_{i+1}-sub-path of P could be replaced by the arc (t_i, t_{i+1}) to create a cheaper path. Once the arc (t_i, t_{i+1}) has been saturated, the SSP algorithm cannot augment along type-1-paths anymore. Therefore, the SSP algorithm will augment along all type-1-paths that it encounters before it augments along all type-3-paths that it encounters.

Since during the time the SSP algorithm augments along type-1-paths no other augmentations alter the part of the residual network corresponding to G_i, the corresponding sub-paths P' are paths in G_i that the SSP algorithm encounters when run on the network G_i. Using the induction hypothesis, this yields that all type-1-paths that the SSP algorithm encounters have costs from the interval

$$[0 + 7 + 0, 1 + \big(2^{i+3} - 5\big) + 1] = [7, 2^{i+3} - 3] \subseteq [7, 2^{i+4} - 5]\,.$$

Since all of these type-1-paths have less costs than the two type-2-paths, the SSP algorithm will augment along them as long as there still exists an augmenting s_i-t_i-sub-path P'. Due to the choice of capacities this is the case until both arcs (s_{i+1}, s_i) and (t_i, t_{i+1}) are saturated. Therefore, the SSP algorithm will not augment along any type-2-path.

When analyzing the costs of type-3-paths, we have to look at the t_i-s_i-sub-paths. Let ℓ be the number of s_i-t_i-paths that the SSP algorithm encounters when run on the network G_i and let $P_1, P_2, \dots, P_\ell$ be the corresponding paths in the same order, in which they were encountered. Then Lemma 5.1.3 yields that for any $j \in \{1, \dots, \ell\}$ after augmenting along the paths $P_1, P_2, \dots, P_j$ the cheapest t_i-s_i-path in the residual network is $\overleftarrow{P_j}$. Property 4.2.4 yields that it is the only cheapest path. Also, the residual network we obtain if we then augment via $\overleftarrow{P_j}$ is equal to the residual network obtained, when only augmenting along the paths $P_1, P_2, \dots, P_{j-1}$. Starting with $j = \ell$ this yields that the t_i-s_i-sub-paths corresponding to the type-3-paths that the SSP algorithm encounters are equal to $\overleftarrow{P_\ell}, \dots, \overleftarrow{P_1}$. By induction the costs of each such path P_j lie in $[7, 2^{i+3} - 5]$. This yields that every type-3-path that the SSP algorithm encounters has costs from the interval

$$\begin{aligned}
&[(2^{i+3} - 1) - (2^{i+3} - 5) + (2^{i+3} - 1), (2^{i+3} + 1) - 7 + (2^{i+3} + 1)] \\
&= [2^{i+3} + 3, 2^{i+4} - 5] \subseteq [7, 2^{i+4} - 5] .
\end{aligned}$$

The previous argument also shows that the SSP algorithm encounters on G_{i+1} twice as many paths as on G_i because it encounters ℓ type-1-paths, no type-2-path, and ℓ type-3-paths, where ℓ denotes the number of paths that the SSP algorithm encounters on G_i. □

Since the SSP algorithm augments along m paths when run on the network G_1, it will augment along $2^{i-1} \cdot m$ paths when run on the network G_i. Note, that at the end of the SSP algorithm, when run on G_i for $i > 1$, only the 4 arcs incident to s_i and t_i carry flow.

Construction of G from G_k. Let $N_k = 2^{k-1} \cdot m$, which is the value of a maximum s_k-t_k flow in G_k. We will now use G_k to define $G = (V, E)$ as follows (see also Figure 5.3).

- $V := V_k \cup A \cup B \cup C \cup D \cup \{s, t\}$, with $A := \{a_1, \dots, a_M\}$, $B := \{b_1, \dots, b_M\}$, $C := \{c_1, \dots, c_M\}$, and $D := \{d_1, \dots, d_M\}$. $E := E_k \cup E_a \cup E_b \cup E_c \cup E_d$.

- E_a contains the edges (a_i, a_{i-1}), $i \in \{2, \dots, M\}$, with cost interval $[2^{k+5} - 1, 2^{k+5}]$ and infinite capacity, (s, a_i), $i \in \{1, \dots, M\}$, with cost interval $[0, 1]$ and capacity N_k, and (a_1, s_k) with cost interval $[2^{k+4} - 1, 2^{k+4}]$ and infinite capacity.

- E_b contains the edges (b_i, b_{i-1}), $i \in \{2, \dots, M\}$, with cost interval $[2^{k+5} - 1, 2^{k+5}]$ and infinite capacity, (s, b_i), $i \in \{1, \dots, M\}$, with cost interval $[0, 1]$ and capacity N_k, and (b_1, t_k) with cost interval $[2^{k+5} - 1, 2^{k+5}]$ and infinite capacity.

- E_c contains the edges (c_{i-1}, c_i), $i \in \{2, \dots, M\}$, with cost interval $[2^{k+5} - 1, 2^{k+5}]$ and infinite capacity, (c_i, t), $i \in \{1, \dots, M\}$, with cost interval $[0, 1]$ and capacity N_k, and (s_k, c_1) with cost interval $[2^{k+5} - 1, 2^{k+5}]$ and infinite capacity.

- E_d contains the edges (d_{i-1}, d_i), $i \in \{2, \dots, M\}$, with cost interval $[2^{k+5} - 1, 2^{k+5}]$ and infinite capacity, (d_i, t), $i \in \{1, \dots, m\}$, with cost interval $[0, 1]$ and capacity N_k, and (t_k, d_1) with cost interval $[2^{k+4} - 1, 2^{k+4}]$ and infinite capacity.

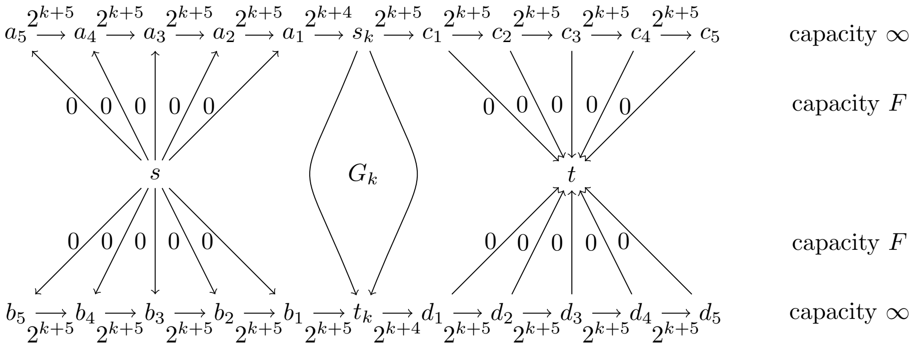


Figure 5.3: G with G_k as sub-graph with approximate edge costs on the edges.

Theorem 5.1.6. *The SSP algorithm encounters $m \cdot 2^{k-1} \cdot 2M$ paths on the network G.*

Proof. We categorize the different s-t-paths that the SSP algorithm encounters on G by the node after s and the node before t. Each such s-t-path can be described as an $\{a_i, c_j\}$-, $\{a_i, d_j\}$-, $\{b_i, c_j\}$-, or $\{b_i, d_j\}$-path for some $i, j \in \{1, \dots, M\}$.

All s_k-t_k-paths encountered by the SSP algorithm, when run on G_k, have costs from the interval $[7, 2^{k+3} - 5]$ in accordance with Lemma 5.1.4. For any $i \in \{1, \dots, m\}$, the costs of the s-a_i-s_k-path and the t_k-d_i-t-path lie in $[\alpha_i, \alpha_i + (i+1)]$ with $\alpha_i = 2^{k+5} i - 2^{k+4} - i$ and the costs of the s-b_i-t_k-path and the s_k-c_i-t-path lie in $[\beta_i, \beta_i + (i+1)]$ with $\beta_i = 2^{k+5} i - i$. Furthermore $i < M + 1 < 2^{k+3}$.

Therefore, the SSP algorithm will only augment along $\{a_i, c_j\}$-paths if no $\{a_i, d_j\}$-paths are available. Also, any $\{a_i, d_i\}$-path is shorter than any $\{b_i, c_i\}$-path and any $\{b_i, c_i\}$-path is shorter than any $\{a_{i+1}, d_{i+1}\}$-path. Finally, any $\{b_i, c_j\}$-path is shorter than any $\{a_{i+1}, c_j\}$-path or $\{b_i, d_{j+1}\}$-path. Therefore, the SSP algorithm will start augmenting along $\{a_1, d_1\}$-paths. After augmenting along $\{a_i, d_i\}$-paths it will augment along $\{b_i, c_i\}$-paths and after augmenting along $\{b_i, c_i\}$-paths it will augment along $\{a_{i+1}, d_{i+1}\}$-paths. Due to the choice of the capacities we can see that once the SSP algorithm starts augmenting along an $\{a_i, d_i\}$-path it keeps augmenting along $\{a_i, d_i\}$-paths until there is no s_k-t_k-path in the residual network that lies completely in the sub-network corresponding to G_k. Also, once the SSP algorithm starts augmenting along an $\{b_i, c_i\}$-path it keeps augmenting along $\{b_i, c_i\}$-paths until there is no t_k-s_k-path in the residual network that lies completely in the sub-network corresponding to G_k. After the SSP algorithm augmented along the last $\{a_i, d_i\}$-path the residual network in the sub-network corresponding to G_k is equal to the residual network of a maximum flow in G_k. After the SSP algorithm augmented along the last $\{b_i, c_i\}$-path the residual network in the sub-network corresponding to G_k is equal to G_k. We can see that the SSP algorithm augments along an $\{a_i, d_i\}$-path for every path P it encounters on G_k and along an $\{b_i, c_i\}$-path for the reverse path $\overleftarrow{P}$ of every path P it encounters on G_k. Therefore, the SSP-algorithm will augment M times along paths corresponding to the paths it encounters on G_k and M times along paths corresponding to the reverse paths of these paths and therefore augment along $2M$ times as many paths in G as in G_k. □

We now complete the proof of Theorem 5.1.1 using Theorem 5.1.6 and the definitions of k and M.

Proof of Theorem 5.1.1. To show that G contains $2n + 2k + 2 + 4M$ nodes and $m + 2n + 4k - 4 + 8M$ edges, we observe that G_1 has $2n + 2$ nodes and $m + 2n$ edges, the $k - 1$ iterations to create G_k add a total of $2k - 2$ nodes and $4k - 4$ edges and the construction of G from G_k adds $4M + 2$ nodes and $8M$ edges. This gives a total of $2n + 2 + 2k - 2 + 4M + 2 = 2n + 2k + 2 + 4M$ nodes and $m + 2n + 4k - 4 + 8M$ edges. Since $k, M = O(n)$ and $m \geq n$, G has $O(n)$ nodes and $O(m)$ edges and forces the SSP algorithm to encounter $m \cdot 2^{k-1} \cdot 2M = \Omega(m\phi M) = \Omega(m \cdot \phi \cdot \min\{n, \phi\})$ paths on G. □

For $\phi = \Omega(n)$ this lower bound shows that the upper bound of $O(mn\phi)$ augmentation steps in Theorem 4.2.2 is tight.

5.2 Minimum-Mean Cycle Canceling Algorithm

In this section we show a lower bound on the number of iterations that the minimum-mean cycle canceling (MMCC) algorithm requires in the setting of smoothed analysis. For an introduction to the MMCC algorithm we refer to Section 4.3.1. In Section 5.2.1 we prove the following result.

Theorem 5.2.1. *For every n, every $m \in \{n, n+1, \ldots, n^2\}$, and every $64 \leq \phi \leq 2^n$, there exists an instance with $\Theta(n)$ nodes and $\Theta(m)$ edges for which the MMCC* *algorithm requires $\Omega(m\log(\phi))$ iterations, independent of the realization of the edge costs.*

Note that Theorem 5.2.1 only holds for $\phi \geq 64$. We did not try to optimize this bound on ϕ, but some lower bound on ϕ is necessary to show that our result holds independent of the realization of the edge costs. This is because if we have $\phi = 1$ (that is, all edge costs are drawn uniformly at random from the interval $[0,1]$), there exist realizations of the edge costs for which the starting flow is already optimal and no cycles have to be canceled. We do believe that even in the case $\phi = 1$, instances can be constructed for which the expected number of iterations required by the MMCC algorithm is $\Omega(m)$.

If ϕ is sufficiently large, namely $\phi = \Omega(n^2)$, we can improve our lower bound. In Section 5.2.2 we prove the following result.

Theorem 5.2.2. *For every $n \geq 4$ and every $m \in \{n, n+1, \ldots, n^2\}$, there exists an instance with $\Theta(n)$ nodes and $\Theta(m)$ edges, and $\phi = \Theta(n^2)$, for which the MMCC algorithm requires $\Omega(mn)$ iterations, independent of the realization of the edge costs.*

Note that this is indeed a stronger lower bound than the bound for general ϕ, since we have $m\log(\phi) = \Theta(m\log(n))$ for $\phi = \Theta(n^2)$.

5.2.1 General Lower Bound

In this section we prove Theorem 5.2.1. We describe a construction that, for every n, every $m \in \{n, n+1, \ldots, n^2\}$, and every $\phi \leq 2^n$, provides an instance with $\Theta(n)$ nodes and $\Theta(m)$ edges for which the MMCC algorithm requires $\Omega(m\log(\phi))$ iterations. For simplicity we describe the initial residual network G_{f^0}, which occurs after a flow f^0 satisfying all the budgets has been found, but before the first minimum-mean cycle has been canceled. For completeness, we will explain at the end of the description of G_{f^0} how to choose the network G, the budgets, and the starting flow f^0 such that G_{f^0} is the initial residual network.

We now describe how to construct G_{f^0} given n, m, and ϕ. As mentioned at the start of Chapter 4, all logarithms are to the base 2. We define $k_w = \lfloor \frac{1}{2}(\log(\phi) - 4) \rfloor$ and $k_x = \lfloor \frac{1}{2}(\log(\phi) - 5) \rfloor$. Note that this implies that either $k_x = k_w$, or $k_x = k_w - 1$. For the edge costs we define intervals from which the edge costs are drawn uniformly at random. We define $G_{f^0} = (\mathcal{V}, \mathcal{E})$ as follows (see Figure 5.4).

- $\mathcal{V} = \{a, b, c, d\} \cup U \cup V \cup W \cup X$, where $U = \{u_1, \ldots, u_n\}$, $V = \{v_1, \ldots, v_n\}$, $W = \{w_1, \ldots, w_{k_w}\}$, and $X = \{x_1, \ldots, x_{k_x}\}$.
- $\mathcal{E} = E_{uv} \cup E_a \cup E_b \cup E_c \cup E_d \cup E_w \cup E_x$.
- E_{uv} is an arbitrary subset of $U \times V$ of cardinality m. Each arc (u_i, v_j) has capacity 1 and cost interval $[0, 1/\phi]$.
- E_a contains the arcs (a, u_i), E_b contains the arcs (u_i, b), E_c contains the arcs (c, v_i), and E_d contains the arcs (v_i, d) $(i = 1, \ldots, n)$. All these arcs have infinite capacity and cost interval $[0, 1/\phi]$.

- E_w contains the arcs (d, w_i) and (w_i, a) $(i = 1, \dots, k_w)$. An arc (d, w_i) has capacity m and cost interval $[0, 1/\phi]$. An arc (w_i, a) has capacity m and cost interval $[-2^{2-2i}, -2^{2-2i} + 1/\phi]$.
- E_x contains the arcs (b, x_i) and (x_i, c) $(i = 1, \dots, k_x)$. An arc (b, x_i) has capacity m and cost interval $[0, 1/\phi]$. An arc (x_i, c) has capacity m and cost interval $[-2^{1-2i}, -2^{1-2i} + 1/\phi]$.

Note that all cost intervals have width $1/\phi$ and therefore correspond to valid probability densities for the edge costs, since the costs are drawn uniformly at random from these intervals. The arcs (w_i, a) and (x_i, c) have a cost interval that corresponds to negative arc costs. The residual network G_{f^0} with these negative arc costs can be obtained by having the following original instance G (before computing a flow satisfying the budget requirements): All nodes, edges, costs and capacities are the same as in G_{f^0}, except that instead of the arcs (w_i, a) we have edges (a, w_i) with capacity m and cost interval $[2^{2-2i} - 1/\phi, 2^{2-2i}]$ and instead of the arcs (x_i, c) we have edges (c, x_i) with capacity m and cost interval $[2^{1-2i} - 1/\phi, 2^{1-2i}]$. In addition, node a has budget $k_w m$, node c has budget $k_x m$, nodes w_i and x_i have budget $-m$ and all other nodes have budget 0. If we now choose as the initial feasible flow the flow f^0 that sends m units from a to each node w_i and from c to each node x_i, then we obtain the initial residual network G_{f^0}.

We now show that the MMCC algorithm needs $\Omega(m \log(\phi))$ iterations for initial residual network G_{f^0}. First we make some basic observations. A minimum-mean cycle C never contains the path $P_j = (d, w_j, a)$ if the path $P_i = (d, w_i, a)$ has positive residual capacity for some $i < j$, since the mean costs of C can be improved by substituting P_j by P_i in C. Analogously, C never contains the path $P_j = (b, x_j, c)$ if the path $P_i = (b, x_i, c)$ has positive residual capacity for some $i < j$. Also, since all cycles considered have mean costs strictly less than $-1/\phi$, the mean costs of a cycle do not decrease when an extra arc with costs at least $-1/\phi$ is added to the cycle. In addition, since the arcs (w_i, a) and (x_i, c) are saturated in the order cheapest to most expensive, none of these arcs will ever be included in reverse direction in the minimum-mean cycle. The above observations lead to three kinds of candidates for the minimum-mean cycle: cycles (d, w_i, a, u, v, d), cycles (b, x_i, c, v, u, b), and cycles $(d, w_i, a, u, b, x_j, c, v, d)$. Here u and v are arbitrary nodes in U and V, respectively. In the following series of lemmas we compare the mean costs of these cycles. Here u and v are again arbitrary nodes in U and V, possibly different for the cycles that are compared. In our computations we always assume worst-case realization of the edge costs, that is, if we want to show that a cycle C_1 has lower mean costs than a cycle C_2, we assume that all arcs in C_1 take the highest costs in their cost interval, while all arcs in C_2 take the lowest costs in their cost interval (an arc that appears in both C_1 and C_2 can even take its highest costs in C_1 and its lowest costs in C_2 in the analysis).

Lemma 5.2.3. *The cycle $C_1 = (d, w_i, a, u, v, d)$ has lower mean costs than the cycle $C_2 = (b, x_i, c, v, u, b)$.*

Proof. Since the cycles have equal length, we can compare their total costs instead of their mean costs. We have

$$\begin{aligned} c(C_1) - c(C_2) &\leq \left(-2^{2-2i} + 5/\phi\right) - \left(-2^{1-2i} - 1/\phi\right) \\ &\leq -64/\phi + 6/\phi < 0 \end{aligned}$$

Here the second inequality holds since $i \leq k_x \leq \frac{1}{2}(\log(\phi) - 5)$. □

Lemma 5.2.4. *The cycle $C_1 = (b, x_i, c, v, u, b)$ has lower mean costs than the cycle $C_2 = (d, w_{i+1}, a, u, v, d)$.*

Proof. Since the cycles have equal length, we can compare their total costs instead of their mean costs. We have

$$\begin{aligned} c(C_1) - c(C_2) &\leq \left(-2^{1-2i} + 4/\phi\right) - \left(-2^{2-2(i+1)}\right) \\ &\leq -64/\phi + 4/\phi < 0 \end{aligned}$$

Here the second inequality holds since $i + 1 \leq k_w \leq \frac{1}{2}(\log(\phi) - 4)$. □

Lemma 5.2.5. *The cycle $C_1 = (d, w_i, a, u, v, d)$ has lower mean costs than the cycle $C_2 = (d, w_i, a, u, b, x_i, c, v, d)$.*

Proof. We have

$$\begin{aligned} \frac{c(C_1)}{|C_1|} - \frac{c(C_2)}{|C_2|} &\leq \frac{\left(-2^{2-2i} + 5/\phi\right)}{5} - \frac{\left(-2^{2-2i} - 2^{1-2i}\right)}{8} \\ &\leq \frac{-8}{5\phi} + \frac{1}{\phi} < 0 \end{aligned}$$

Here the second inequality holds since $i \leq k_x \leq \frac{1}{2}(\log(\phi) - 5)$. □

Lemma 5.2.6. *The cycle $C_1 = (b, x_i, c, v, u, b)$ has lower mean costs than the cycle $C_2 = (b, x_i, c, v, d, w_{i+1}, a, u, b)$.*

Proof. We have

$$\begin{aligned} \frac{c(C_1)}{|C_1|} - \frac{c(C_2)}{|C_2|} &\leq \frac{\left(-2^{1-2i} + 4/\phi\right)}{5} - \frac{\left(-2^{1-2i} - 2^{2-2(i+1)}\right)}{8} \\ &\leq \frac{-8}{5\phi} + \frac{4}{5\phi} < 0 \end{aligned}$$

Here the second inequality holds since $i + 1 \leq k_w \leq \frac{1}{2}(\log(\phi) - 4)$. □

The above observations and lemmas allow us to determine the number of iterations that the MMCC algorithm requires for initial residual network G_{f^0}.

Theorem 5.2.7. *The MMCC algorithm requires $m(k_w + k_x)$ iterations for initial residual network G_{f^0}, independent of the realization of the edge costs.*

Proof. For the first iteration only cycles (d, w_i, a, u, v, d) and $(d, w_i, a, u, b, x_i, c, v, d)$ are available. According to Lemma 5.2.5, all cycles (d, w_i, a, u, v, d) have lower mean costs than all cycles $(d, w_i, a, u, b, x_i, c, v, d)$ and therefore the first iteration will augment along a cycle (d, w_1, a, u, v, d). After the first iteration, an arc from V to U will become available, and therefore cycles (b, x_i, c, v, u, b). According to Lemma 5.2.3 these cycles have higher mean costs than cycles (d, w_i, a, u, v, d) however, and therefore in the first m iterations the MMCC algorithm will augment along cycles (d, w_1, a, u, v, d).

After the first m iterations, the arc (d, w_1), the arc (w_1, a), and all arcs in E_{uv} will be saturated. The available cycles are now (b, x_i, c, v, u, b) and $(b, x_i, c, v, d, w_{i+1}, a, u, b)$. According to Lemma 5.2.6, all cycles (b, x_i, c, v, u, b) have lower mean costs than all cycles $(b, x_i, c, v, d, w_{i+1}, a, u, b)$. The next iteration will therefore augment along a cycle (b, x_1, c, v, u, b). After this iteration, an arc from U to V becomes available and therefore a cycle (d, w_2, a, u, v, d), but according to Lemma 5.2.4 all cycles (b, x_i, c, v, u, b) have lower mean costs than cycles (d, w_{i+1}, a, u, v, d) and therefore in iterations $m + 1, \ldots, 2m$ the MMCC algorithm augments along cycles (b, x_1, c, v, u, b).

After $2m$ iterations, we again obtain G_{f^0}, except that now arcs (d, w_1), (w_1, a), (b, x_1), and (x_1, c) are saturated and that there is some flow on the infinite capacity arcs (a, u_i), (u_i, b), (c, v_i), and (v_i, d). The MMCC algorithm will keep augmenting among m cycles (d, w_i, a, u, v, d) followed by m cycles (b, x_i, c, v, u, b) until all arcs (w_i, a) and (x_i, c) are saturated and no negative-cost cycles remain. The total number of iterations the MMCC algorithm needs is therefore $m(k_w + k_x)$. $\square$

The instance G, initial flow f^0, and Theorem 5.2.7 allow us to complete the proof of Theorem 5.2.1.

Proof of Theorem 5.2.1. Follows directly from the instance G, initial flow f^0, Theorem 5.2.7, and the definitions of k_w and k_x. $\square$

5.2.2 Lower Bound for ϕ Dependent on n

This section is devoted to the proof of Theorem 5.2.2. In Section 5.2.1 we considered the setting where ϕ is arbitrary. In this setting we showed that the MMCC algorithm needs $\Omega(m \log(\phi))$ iterations. We can improve the lower bound if ϕ is much larger than n. In this section we consider the case where $\phi = \Omega(n^2)$. In particular, we describe a construction that for every $n \geq 4$ and every $m \in \{n, \ldots, n^2\}$ provides an instance with $\Theta(n)$ nodes, $\Theta(m)$ edges, and $\phi = \Theta(n^2)$ for which the MMCC algorithm needs $\Omega(mn)$ iterations.

As in Section 5.2.1, we construct a flow network H and a feasible flow f^0 on H, which together provide an initial residual network H_{f^0}. The residual network H_{f^0} that we use to show our bound is very similar to the initial residual network G_{f^0} that was used to show the bound for general ϕ in Section 5.2.1. Below we describe the differences (see Figure 5.5 for an illustration). We set $\phi = 400000n^2$. The constant

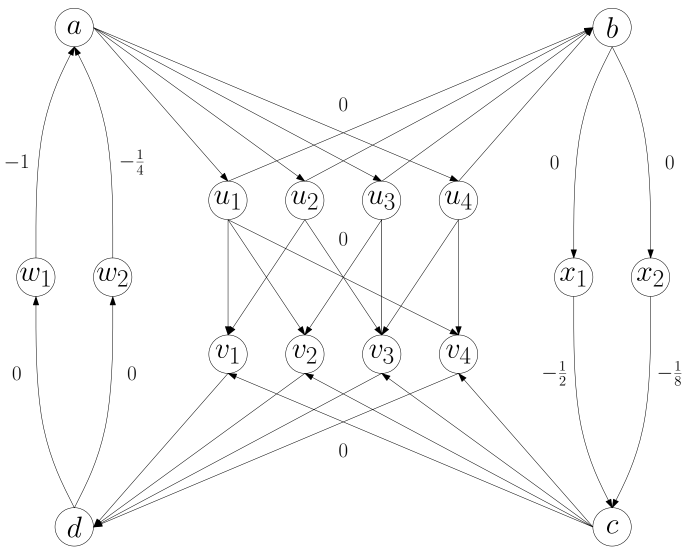


Figure 5.4: The initial residual network G_{f^0} for which the MMCC algorithm needs $O(m \log(\phi))$ iterations for $n = 4$, $m = 9$, and $\phi = 64$. Next to the arcs are the approximate arc costs.

of 400000 is large, but for the sake of readability and ease of calculations we did not try to optimize it.

- The node set W now consists of n nodes $\{w_1, \dots, w_n\}$ and the node set X now consists of n nodes $\{x_1, \dots, x_n\}$.
- Node a is split into two nodes a_1 and a_2. From node a_1 to a_2 there is a directed path consisting of n arcs, all with infinite capacity and cost interval $[0, 1/\phi]$. arcs (a, u_i) are replaced by arcs (a_2, u_i) with infinite capacity and cost interval $[0, 1/\phi]$. arcs (w_i, a) are replaced by arcs (w_i, a_1) with capacity m and cost interval $[-(\frac{n-3}{n})^{2i-2}, -(\frac{n-3}{n})^{2i-2} + \frac{1}{\phi}]$.
- Node c is split into two nodes c_1 and c_2. From node c_1 to c_2 there is a directed path consisting of n arcs, all with infinite capacity and cost interval $[0, 1/\phi]$. arcs (c, v_i) are replaced by arcs (c_2, v_i) with infinite capacity and cost interval $[0, 1/\phi]$. arcs (x_i, c) are replaced by arcs (x_i, c_1) with capacity m and cost interval $[-(\frac{n-3}{n})^{2i-1}, -(\frac{n-3}{n})^{2i-1} + \frac{1}{\phi}]$.

Note that this is a valid choice of cost intervals for the arcs (w_i, a_1) and (x_i, c_1) and that they all have negative costs, since (x_n, c_1) is the most expensive of them and we have

$$-\left(\frac{n-3}{n}\right)^{2n-1} + \frac{1}{\phi} \leq -\left(1 - \frac{3}{n}\right)^{2n} + \frac{1}{400000n^2} \leq -(e^{-6})^2 + \frac{1}{6400000} < 0,$$

where the second inequality follows from $-\left(1-\frac{3}{n}\right)^n \leq -e^{-6}$ for $n \geq 4$.

As in Section 5.2.1, there are three kinds of candidate cycles for the minimum-mean cost cycle: cycles (d, w_i, a, u, v, d), cycles (b, x_i, c, v, u, b), and cycles $(d, w_i, a, u, b, x_j, c, v, d)$. Here u and v are arbitrary nodes in U and V, respectively. Also, a stands for the path from a_1 to a_2 and c stands for the path from c_1 to c_2. Again we assume worst-case realizations of the edge costs and compare the mean costs of the cycles in a series of lemmas.

Lemma 5.2.8. *The cycle $C_1 = (d, w_i, a, u, v, d)$ has lower mean costs than the cycle $C_2 = (b, x_i, c, v, u, b)$.*

Proof. Since the cycles have equal length, we can compare their total costs instead of their mean costs. We have

$$\begin{aligned} c(C_1) - c(C_2) &\leq \left(-\left(\frac{n-3}{n}\right)^{2i-2} + \frac{n+5}{\phi}\right) - \left(-\left(\frac{n-3}{n}\right)^{2i-1} - \frac{1}{\phi}\right) \\ &\leq -\frac{3e^{-12}}{n} + \frac{n+6}{\phi} < 0 \end{aligned}$$

Here the second inequality holds since $i \leq n$ and $-\left(\frac{n-3}{n}\right)^{2n} \leq -e^{-12}$ for $n \geq 4$. □

Lemma 5.2.9. *The cycle $C_1 = (b, x_i, c, v, u, b)$ has lower mean costs than the cycle $C_2 = (d, w_{i+1}, a, u, v, d)$.*

Proof. Since the cycles have equal length, we can compare their total costs instead of their mean costs. We have

$$\begin{aligned} c(C_1) - c(C_2) &\leq \left(-\left(\frac{n-3}{n}\right)^{2i-1} + \frac{n+4}{\phi}\right) - \left(-\left(\frac{n-3}{n}\right)^{2(i+1)-2}\right) \\ &\leq -\frac{3e^{-12}}{n} + \frac{n+4}{\phi} < 0 \end{aligned}$$

Here the second inequality holds since $i+1 \leq n$ and $-\left(\frac{n-3}{n}\right)^{2n} \leq -e^{-12}$ for $n \geq 4$. □

Lemma 5.2.10. *The cycle $C_1 = (d, w_i, a, u, v, d)$ has lower mean costs than the cycle $C_2 = (d, w_i, a, u, b, x_i, c, v, d)$.*

Proof. We have

$$\begin{aligned} \frac{c(C_1)}{|C_1|} - \frac{c(C_2)}{|C_2|} &\leq \frac{\left(-\left(\frac{n-3}{n}\right)^{2i-2} + \frac{n+5}{\phi}\right)}{n+5} - \frac{\left(-\left(\frac{n-3}{n}\right)^{2i-2} - \left(\frac{n-3}{n}\right)^{2i-1}\right)}{2n+8} \\ &\leq -e^{-12}\left(\frac{n+15}{(n+5)(2n+8)n}\right) + \frac{1}{\phi} < 0 \end{aligned}$$

Here the second inequality holds since $i \leq n$ and $-\left(\frac{n-3}{n}\right)^{2n} \leq -e^{-12}$ for $n \geq 4$. □

Lemma 5.2.11. *The cycle $C_1 = (b, x_i, c, v, u, b)$ has lower mean costs than the cycle $C_2 = (b, x_i, c, v, d, w_{i+1}, a, u, b)$.*

Proof. We have

$$\begin{aligned}\frac{c(C_1)}{|C_1|} - \frac{c(C_2)}{|C_2|} &\leq \frac{\left(-\left(\frac{n-3}{n}\right)^{2i-1} + \frac{n+4}{\phi}\right)}{n+5} - \frac{\left(-\left(\frac{n-3}{n}\right)^{2i-1} - \left(\frac{n-3}{n}\right)^{2(i+1)-2}\right)}{2n+8} \\ &\leq -e^{-12}\left(\frac{n+15}{(n+5)(2n+8)n}\right) + \frac{n+4}{(n+5)\phi} < 0\end{aligned}$$

Here the second inequality holds since $i + 1 \leq n$ and $-\left(\frac{n-3}{n}\right)^{2n} \leq -e^{-12}$ for $n \geq 4$. □

The above lemmas allow us to determine the number of iterations that the MMCC algorithm requires for initial residual network H_{f^0}.

Theorem 5.2.12. *The MMCC algorithm requires $2mn$ iterations for initial residual network H_{f^0}, independent of the realization of the edge costs.*

Proof. The proof is similar to the proof of Theorem 5.2.7. Because cycles (d, w_i, a, u, v, d) have lower mean costs than both cycles (b, x_i, c, v, u, b) and cycles $(d, w_i, a, u, b, x_i, c, v, d)$ according to Lemma 5.2.8 and Lemma 5.2.10, the first m iterations will augment flow along cycles (d, w_1, a, u, v, d). After these m iterations, arcs (d, w_1) and (w_1, a_1) are saturated and the arcs in E_{uv} have positive residual capacity only in the direction from V to U.

Cycles (b, x_i, c, v, u, b) have lower mean costs than both cycles (d, w_{i+1}, a, u, v, d) and cycles $(b, x_i, c, v, d, w_{i+1}, a, u, b)$ according to Lemma 5.2.9 and Lemma 5.2.11. Thus, the next m iterations will augment flow along cycles (b, x_1, c, v, u, b). After the first $2m$ iterations, the residual network is the same as H_{f^0}, except that arcs (d, w_1), (w_1, a_1), (b, x_1), and (x_1, c_1) are saturated and there is some flow on several arcs of infinite capacity. The MMCC algorithm will keep augmenting along m cycles (d, w_i, a, u, v, d) followed by m cycles (b, x_i, c, v, u, b), until all arcs (w_i, a_1) and (x_i, c_1) are saturated. At this point no negative cycles remain in the residual network and the MMCC algorithm terminates after $2mn$ iterations. □

Initial residual network H_{f^0} and Theorem 5.2.12 allow us to complete the proof of Theorem 5.2.2.

Proof of Theorem 5.2.2. Follows directly from the instance H, initial flow f^0, and Theorem 5.2.12. □

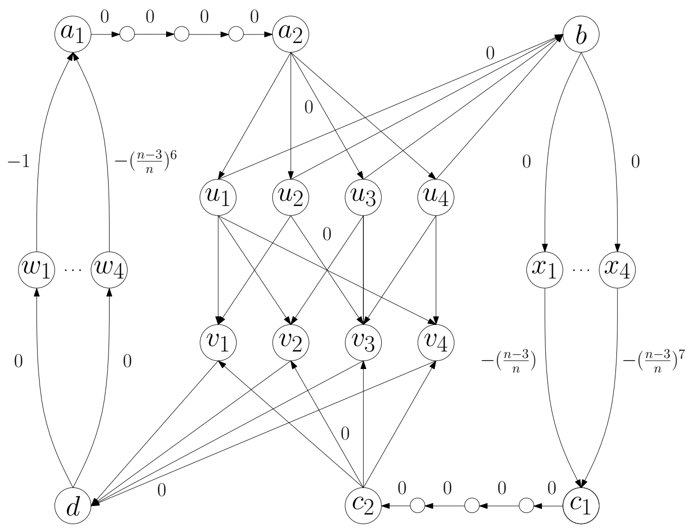


Figure 5.5: The initial residual network H_{f^0} for which the MMCC algorithm needs $\Omega(mn)$ iterations for $n = 4$, $m = 9$, and $\phi = 400000n^2$. Next to the arcs are the approximate arc costs.

5.3 Network Simplex Algorithm

5.3.1 Introduction

In this section we show a lower bound on the number of iterations that the network simplex (NS) algorithm needs in the setting of smoothed analysis. The NS algorithm is an adaptation of the simplex method for linear programming [22] that takes advantage of the special structure of the minimum-cost flow problem. The NS algorithm maintains a spanning tree T of edges on which flow is allowed to be different from 0 or the edge capacity. On all other edges the flow is either equal to 0 or equal to the edge capacity. In every iteration an edge is added to T, creating a unique cycle with negative costs. Flow is maximally augmented along this cycle to decrease the costs of the flow. Because of this flow augmentation at least one arc gets saturated and the edge corresponding to this arc is removed from T, such that T is again a spanning tree. The NS algorithm terminates when no edge can be added to T to form a cycle with negative costs. Before we prove our smoothed lower bound on the number of iterations required by the NS algorithm, we first introduce the algorithm. For a more elaborate introduction to the NS algorithm we refer to Ahuja et al. [1].

Definition of the Algorithm

The NS algorithm starts with an initial spanning tree structure (T^0, L^0, U^0) and associated flow f^0. It maintains such a spanning tree structure (T^i, L^i, U^i) throughout the execution of the algorithm. In each iteration i of the NS algorithm, each edge in E is included in exactly one of T^i, L^i, and U^i, and it holds that

- $f_e^i = 0$ for all edges $e \in L^i$,
- $f_e^i = u_e$ for all edges $e \in U^i$,
- $0 \leq f_e^i \leq u_e$ for all edges $e \in T^i$, and
- the edges in T^i form a spanning tree of G (if we consider the undirected version of both the edges of T^i and the graph G).

If the MCF problem has a feasible solution, such an initial spanning tree structure (T^0, L^0, U^0) can always be found by first finding any feasible flow and then augmenting flow along cycles in the residual network consisting of only arcs that are not saturated, but have a non-zero amount of flow on them, until no such cycles remain. Note that the structure (T^i, L^i, U^i) uniquely determines the flow f^i, since the edges in T^i form a tree. In addition to the spanning tree structure, the NS algorithm also keeps track of a set of node potentials π_v^i for all nodes $v \in V$. The node potentials are defined such that the potential of a specified root node is 0 and that the potential for other nodes is such that the reduced cost $rc_{uv}^i = c_{uv} - \pi_u^i + \pi_v^i$ of an edge (u, v) equals 0 for all edges $(u, v) \in T^i$.

In each iteration $i + 1$, the NS algorithm tries to improve the current flow by adding an edge to T^i that violates its optimality condition. An edge in L^i violates its optimality condition if it has strictly negative reduced cost, while an edge in U^i violates its optimality condition if it has strictly positive reduced cost. One of the edges e that violates its optimality condition is selected, which creates a unique cycle C in $T^i \cup \{e\}$ (if we consider the undirected version of the edges in T^i and edge e). Flow is maximally augmented along C in the direction that decreases the costs of the flow, until one of the arcs $e' \in C$ gets saturated. The new spanning tree T^{i+1} is obtained by adding edge e to T^i and removing the edge corresponding to e'. Next, we update the sets L^i and U^i, the flow, and the node potentials. This completes the iteration. If any edges violating their optimality condition remain, another iteration is performed. One iteration of the NS algorithm is also called a pivot. The edge e that is added to T^i is called the entering edge and the edge e' that leaves T^i is called the leaving edge. Note that in some cases the entering edge can be the same edge as the leaving edge (when the arc corresponding to the entering edge has the smallest residual capacity of all arcs in the cycle C). Also, if one of the arcs in the cycle C has 0 residual capacity, the flow is not changed in that iteration, but the spanning tree T^i still changes. Such an iteration we call degenerate.

Note that in each iteration, there can be multiple edges violating their optimality condition. There are multiple possible pivot rules that determine which edge enters T^i in this case. In our analysis we use the (widely used in practice) pivot rule that selects as the entering edge, from all edges violating their optimality condition, the edge for which the absolute value of its reduced cost $|rc_e^i|$ is maximum. In case multiple edges corresponding to arcs in C are candidates to be the leaving edge, we choose the one that is most convenient for our analysis. We are now ready to define the NS algorithm (Algorithm 4).

Algorithm 4 Network simplex algorithm.

```
1: if a feasible flow exists then
2:    find an initial spanning tree structure $(T^0, L^0, U^0)$ and compute the corre-
      sponding flow $f^0$ and node potentials $\pi_v^0$
3: else
4:    output that no feasible flow exists
5: end if
6: for $i = 1, 2, \dots$ do
7:    if for spanning tree structure $(T^{i-1}, L^{i-1}, U^{i-1})$ no edge violates its optimality
      condition then
8:       output $f^{i-1}$
9:    end if
10:   find edge $e$ that maximally violates its optimality condition and maximally*
      augment flow along the unique cycle $C$ in $T^{i-1} \cup \{e\}$ in the direction that
      decreases the costs of the flow
11:   let $e'$ be the edge corresponding to the arc that gets saturated by augmenting
      along $C$
12:   set $T^i = T^{i-1} \cup \{e\} \setminus \{e'\}$
13:   update $L^i$, $U^i$, $f^i$, and $\pi_v^i$
14: end for
```

* Since the flow f^i must obey all capacity constraints, the flow is increased by the minimum of $\min\{u_e - f_e^{i-1} \mid e \in C \cap E\}$ and $\min\{f_{e^{-1}}^{i-1} \mid e \in C \text{ and } e^{-1} \in E\}$.

Known Bounds on the Number of Iterations

If a strongly feasible spanning tree structure [1] is used, it can be shown that the number of iterations that the NS algorithm needs is finite. However, Zadeh [66] has shown that there exist instances for which the NS algorithm (with the pivot rule stated above) needs an exponential number of iterations. Orlin [44] has developed a strongly polynomial version of the NS algorithm, which uses cost-scaling. However, this algorithm is rarely used in practice and we will not consider it in the rest of this chapter. For a more elaborate discussion of the NS algorithm we refer to Ahuja et al. [1].

Our Results

We show a lower bound on the number of iterations that the NS algorithm requires in the setting of smoothed analysis. In Section 5.3.2 we prove the following result.

Theorem 5.3.1. *For every n, every $m \in \{n, \dots, n^2\}$, and every $64 \le \phi \le 2^n$ there exists a flow network with $\Theta(n)$ nodes and $\Theta(m)$ edges, and an initial spanning* *tree structure for which the network simplex algorithm requires* $\Omega(m \cdot \min\{n, \phi\} \cdot \phi)$ *non-degenerate iterations with probability* 1*.*

As for our lower bound on the smoothed number of iterations required by the MMCC algorithm (Theorem 5.2.1), we require that ϕ is sufficiently large in Theorem 5.3.1. Again, this is because we show our result independent of the realization of the edge costs, and in the average case ($\phi = 1$) there are realizations of the edge costs for which the starting flow is already optimal.

Note that our lower bound for the NS algorithm is the same as our lower bound on the smoothed number of iterations of the SSP algorithm (Section 5.1). This is no coincidence, since we use essentially the same instance (with some minor changes) to show our lower bound. We show that with the proper choice of the initial spanning tree structure for the NS algorithm, we can ensure that the NS algorithm performs the same flow augmentations as the SSP algorithm and therefore needs the same number of iterations (plus some degenerate ones).

5.3.2 Proof of the Lower Bound

In this section we prove Theorem 5.3.1. The instance H of the minimum-cost flow problem that we use to show this lower bound is very similar to the instance G that we used in Section 5.1 to show a lower bound on the number of iterations that the successive shortest path algorithm needs in the smoothed setting. The differences are that we add an extra path from node s to node t and that the budgets of the nodes are defined slightly differently. For the construction of G and the definition of the parameters k and M, we refer to Section 5.1. In the following we describe how to construct H, using G as a sub-graph. As in Section 5.1, we consider scaled edge cost density functions $g_e \colon [0, \phi] \to [0, 1]$. In the analysis, we assume that all paths from s to t have pairwise different costs, which holds with probability 1, since the edge costs are drawn from continuous probability distributions.

Let V and E be the node and edge sets of flow network G, respectively, as defined in Section 5.1. We construct a flow network $H = (V_H, E_H) = (V \cup Q, E \cup E_q)$ by adding to G a set of nodes Q and a set of edges E_q. Here,

- $Q := \{q_1, q_2, \ldots, q_{2M}\}$.
- E_q contains the edges (q_{i-1}, q_i), $i \in \{2, \ldots, 2M\}$, with cost interval $[2^{k+5} - 1, 2^{k+5}]$ and infinite capacity; (s, q_1), with cost interval $[2^{k+5} - 1, 2^{k+5}]$ and infinite capacity; and (q_{2M}, t) with cost interval $[2^{k+5} - 1, 2^{k+5}]$ and infinite capacity.

The budgets of all nodes are 0, except for node s and t, which have budgets $b(s) = 2MN_k$ and $b(t) = -2MN_k$. Here N_k is the value of a maximum $s_k - t_k$ flow, as defined in Section 5.1. We choose as the initial spanning tree T^0 for the NS algorithm the edges

- (s_1, u_i) $(i = 1, \ldots, n)$ and (w_i, t_1) $(i = 1, \ldots, n)$,

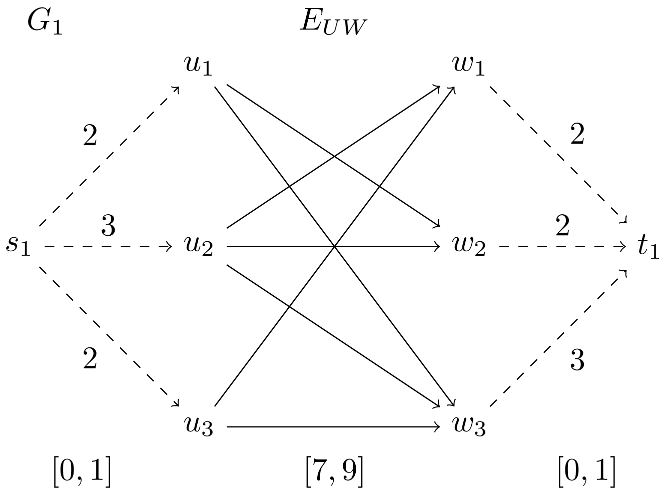


Figure 5.6: Example for G_1 with $n = 3$ and $m = 7$ with capacities different from 1 shown next to the edges and the cost intervals shown below each edge set. Dashed edges are in the initial spanning tree for the network simplex algorithm, while solid edges are not.

- (s_{i+1}, s_i) $(i = 1, \dots, k-1)$ and (t_i, t_{i+1}) $(i = 1, \dots, k-1)$,
- (s, a_1), (a_i, a_{i-1}) $(i = 2, \dots, M)$, and (a_1, s_k),
- (s_k, c_1) and (c_{i-1}, c_i) $(i = 2, \dots, M)$,
- (b_i, b_{i-1}) $(i = 2, \dots, M)$ and (b_1, t_k),
- (d_1, t), (t_k, d_1), and (d_{i-1}, d_i) $(i = 2, \dots, M)$,
- (s, q_1), (q_i, q_{i+1}) $(i = 1, \dots, 2M-1)$, and (q_{2M}, t).

In addition, we define $L^0 = E_H \setminus T^0$ and $U^0 = \emptyset$. The spanning tree structure (T^0, L^0, U^0) corresponds to the flow that sends $2MN_k$ units of flow on the path $(s, q_1, \dots, q_{2M}, t)$ and does not send any flow on other edges. For an illustration of the subnetworks G_1 and G_{i+1}, and the complete network H, we refer to Figures 5.6, 5.7, and 5.8. In these figures, the edges that are in the initial spanning tree T^0 are drawn dashed.

To prove our lower bound on the number of iterations that the NS algorithm needs for flow network H, we link the non-degenerate iterations of the NS algorithm to the iterations of the SSP algorithm for flow network G. According to Theorem 5.1.6, the SSP algorithm needs $\Omega(m \cdot \min\{n, \phi\} \cdot \phi)$ iterations for flow network G. We show that each non-degenerate iteration of the NS algorithm on H corresponds with an iteration of the SSP algorithm on G and that therefore the NS algorithm needs $\Omega(m \cdot \min\{n, \phi\} \cdot \phi)$ iterations for flow network H as well. In our analysis we use

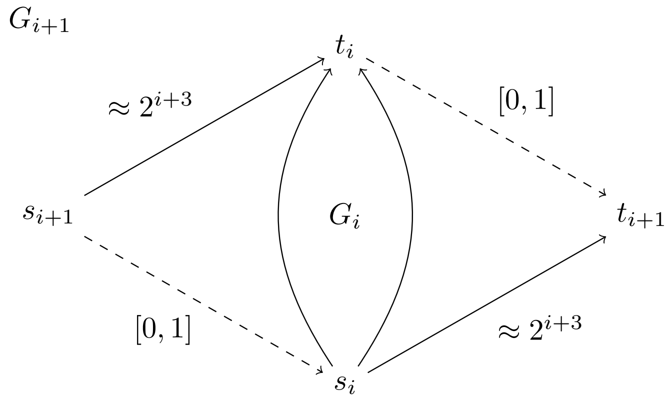


Figure 5.7: G_{i+1} with G_i as a sub-graph with edge costs next to the edges. Dashed edges are in the initial spanning tree for the network simplex algorithm, while solid edges are not.

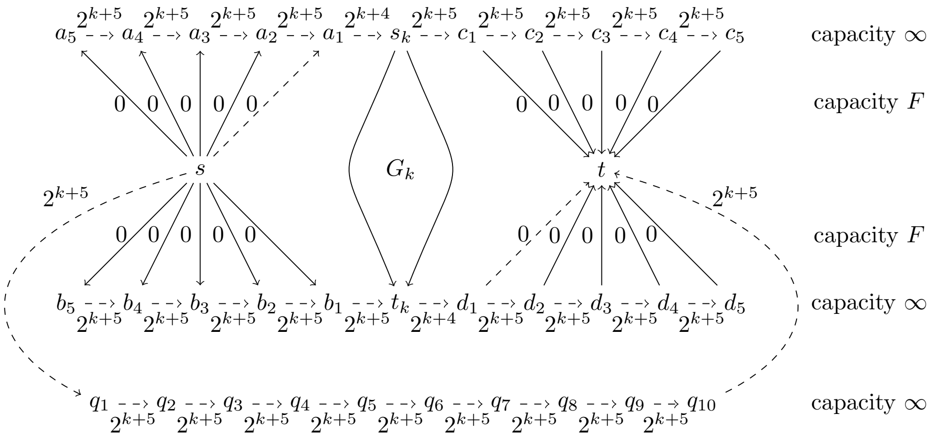

$a_5, a_4, a_3, a_2, a_1, s_k, c_1, c_2, c_3, c_4, c_5$
2^{k+5}, 2^{k+4}
capacity ∞
0
capacity F
s
G_k
t
capacity F
$b_5, b_4, b_3, b_2, b_1, t_k, d_1, d_2, d_3, d_4, d_5$
capacity ∞
$q_1, q_2, q_3, q_4, q_5, q_6, q_7, q_8, q_9, q_{10}$
capacity ∞

Figure 5.8: H with G_k as a sub-graph with approximate edge costs next to the edges. Dashed edges are in the initial spanning tree for the network simplex algorithm, while solid edges are not.

many results on the costs of paths in G from Section 5.1. We will not prove these results again, but refer to the original proofs.

We now first show that path $(s, q_1, \ldots, q_{2M}, t)$ is the most expensive path from s to t in H. Since paths from s_k to t_k have costs less than 2^{k+3} (Lemma 5.1.4), the most expensive $s - t$ paths that do not use the nodes q_i are $(s, a_M, \ldots, c_M, t)$ and $(s, b_M, \ldots, d_M, t)$. Both those paths have the same distribution for their costs. If we arbitrarily choose path $(s, a_M, \ldots, c_M, t)$ and compare its costs with the costs of path $(s, q_1, \dots, q_{2M}, t)$, assuming worst case edge-cost realizations, we have

$$\begin{aligned} & c(s, q_1, \dots, q_{2M}, t) - c(s, a_M, \dots, c_M, t) \\ \geq\ & (2M+1)\left(\frac{2^{k+5}-1}{\phi}\right) - \left(\frac{(2M-1)2^{k+5}+2^{k+4}+2}{\phi}\right) \\ =\ & \frac{3 \cdot 2^{k+4} - 2M - 3}{\phi} > 0, \end{aligned}$$

by the definition of k and M and $\phi \geq 64$. This shows that path $(s, q_1, \dots, q_{2M}, t)$ is the most expensive path from s to t in G.

Since $P_Q = (s, q_1, \dots, q_{2M}, t)$ is the most expensive $s-t$ path, we can obtain a negative cycle K by combining path $\overleftarrow{P_Q}$ with another $s-t$ path P. Clearly, the cheaper P is, the cheaper is K. In addition, if all edges corresponding to the arcs of K except for one edge $(u, v) \in L^i$ are in the current spanning tree T^i, then edge (u, v) has reduced cost equal to the costs of K, since all edges in T^i have reduced cost 0. Similarly, in case $(u, v) \in U^i$, the reduced cost of (u, v) is instead equal to $-K$. Using these two observations, we can conclude that as long as all negative cycles that can be formed using arcs corresponding to edges in the current spanning tree T^i plus one arc that has positive residual capacity in the current residual network consist of path $\overleftarrow{P_Q}$ plus an $s-t$ path P, then the NS algorithm will choose as the entering edge the edge that together with some of the edges in T^i forms the cheapest $s-t$ path.

In Section 5.1 we have shown that the SSP algorithm encounters $2MN_k$ paths on G. Let $P^1, \dots, P^{2MN_k}$ be the paths encountered on G by the SSP algorithm, ordered from cheapest to most expensive. We refer the reader to Section 5.1 for a description of these paths. In the following we show that in the i^{th} non-degenerate iteration of the NS algorithm on H, flow is sent along cycle $\overleftarrow{P_Q} \cup P^i$. We will not provide all the calculations needed to compare the reduced cost of the candidate edges for addition to the spanning tree in each iteration, but it can be checked by tedious computation that the claimed edge is indeed the edge that maximally violates its optimality condition. If flow is sent over one of the edges in E_{UW}, that edge is always a candidate leaving edge, since edges in E_{UW} have capacity 1 and all other capacities are integral. For simplicity we assume that in cases where multiple arcs in the cycle become saturated simultaneously by the flow augmentation, it is always the edge from E_{UW} that leaves the spanning tree.

In the first iteration of the NS algorithm on H, all edges in E_{UW} have negative reduced cost and positive residual capacity, and the edge (u_i, w_j) that together with the initial spanning tree T^0 contains path P^1 is added to T^0, flow is augmented along cycle $\overleftarrow{P_Q} \cup P^1$, and (u_i, w_j) becomes saturated and leaves T^0 again. The spanning tree structure is updated to (T^1, L^1, U^1). For the second iteration, edge (u_i, w_j) is saturated and therefore the edge that together with T^1 contains P^2 will be added to T^1. This will continue for the first m iterations.

At this point, the cheapest $s-t$ path using arcs corresponding to edges in T^m plus one arc with positive residual capacity is the path that is obtained by using either arc (s_2, t_1) or arc (s_1, t_2) (depending on the realization of the edge costs). The next two iterations will therefore be degenerate. In one of these iterations (s_2, t_1) is added to the spanning tree, but edge (t_1, t_2) is saturated, prevents any flow from being sent along the cycle, and is therefore removed from the spanning tree. In the other iteration (s_1, t_2) is added to the spanning tree and (s_2, s_1) is removed. After these two iterations the edges in E_{UW} become eligible for augmenting flow in backward direction and the next m iterations augment flow along the cycles $\overleftarrow{P_Q} \cup P^{m+1}, \dots,$ $\overleftarrow{P_Q} \cup P^{2m}$.

Analogously to the above, every time an edge (s_{i+1}, s_i) gets saturated, two degenerate iterations take place in which edges (s_i, t_{i+1}) and (s_{i+1}, t_i) are added to the spanning tree. This allows flow to be sent through G_i in backward direction, that is, from t_i to s_i. Similarly, every time an arc (t_{i+1}, s_i) (that is, the backward arc corresponding to an original edge (s_i, t_{i+1})) in the residual network gets saturated, two degenerate iterations take place in which edges (t_i, t_{i+1}) and (s_{i+1}, s_i) are added to the spanning tree.

After N_k non-degenerate iterations, there are no paths with positive residual capacity from s_k to t_k. At this point another two degenerate iterations take place. In one of them edge (c_1, t) is added to the spanning tree, but no flow is sent since (s, a_1) has zero residual capacity, and therefore (s, a_1) leaves the spanning tree. In the other degenerate iteration (s, b_1) is added to the spanning tree and (d_i, t) leaves. Now the edges in E_{UW} can be added to the spanning tree again and flow is augmented along them in backward direction during the next m iterations. In particular, in the next iteration flow is augmented along cycle $\overleftarrow{P_Q} \cup P^{N_k+1}$.

Analogously to the above, every time an edge (s, a_i) gets saturated, two degenerate iterations take place. In one of them (c_i, t) enters the spanning tree and (s, a_i) leaves. In the other (s, b_i) enters the spanning tree and (d_1, t) leaves. Also, every time an edge (s, b_i) gets saturated, two degenerate iterations take place. In one of them (d_{i+1}, t) enters the spanning tree and (s, b_i) leaves. In the other (s, a_{i+1}) enters the spanning tree and (c_i, t) leaves.

Finally, after $2MN_k$ non-degenerate iterations none of the edges violate their optimality conditions, and therefore the NS algorithm terminates. From the above discussion we can conclude that the NS algorithm on H requires $2MN_k$ non-degenerate iterations plus several degenerate ones.

Theorem 5.3.2. *For flow network H and initial spanning tree structure (T^0, L^0, U^0), the NS algorithm requires $2MN_k$ non-degenerate iterations with probability 1.*

Proof. Follows immediately from the discussion above. □

We complete the proof of Theorem 5.3.1 using Theorem 5.3.2 and the definitions of M and N_k.

Proof of Theorem 5.3.1. Follows directly from Theorem 5.3.2 and the definitions of M and N_k. □

5.4 Comparison of the Upper and Lower Bounds

In Section 5.3 we have shown a smoothed lower bound of $\Omega(m \cdot \min\{n, \phi\} \cdot \phi)$ for the number of iterations that the NS algorithm needs. This bound is the same as the smoothed lower bound for the SSP algorithm that we have shown in Section 5.1. For the SSP algorithm this lower bound is even tight in case $\phi = \Omega(n)$. Still, the NS algorithm is usually much faster in practice than the SSP algorithm. We believe that the reason for this difference is that the time needed per iteration is much less for the NS algorithm than for the SSP algorithm. In practical implementations of the NS algorithm, the entering edge is usually picked from a small subset (for example of size $\Theta(\sqrt{m})$) of the edges, which removes the necessity of scanning all edges for the edge which maximally violates its optimality condition. Also, the spanning tree structure allows for fast updating of the flow and node potentials, in particular when the flow changes on only a small fraction of the edges. For the SSP algorithm, an iteration consists of finding a shortest path, which takes $O(m + n\log(n))$ time. The experimental results of Kovács [37] seem to support this claim, since on all test instances the SSP algorithm is slower than the NS algorithm, but never more than a factor m. To allow a better comparison of the SSP algorithm and the NS algorithm in the smoothed setting, it would be useful to have a smoothed upper bound on the running-time of the NS algorithm. Finding such an upper bound is our main open problem. Note that the smoothed upper bounds for the running-time of the simplex method for linear programming by Spielman and Teng [56] and by Vershynin [61] do not apply to the NS algorithm. In their models the coefficients and the right-hand side of the constraints of the linear program are perturbed. This is in contrast to our model (Section 4.1), where we perturb the coefficients of the objective function (that is, the edge costs), but not the coefficients and the right-hand side of the constraints. We make this choice because perturbing the coefficients and right-hand side of the constraints would change the structure of the problem. Flows that are feasible for the original problem could be infeasible for the perturbed problem, and vice versa. It might even be the case that the original problem is feasible, but the perturbed problem does not have any feasible solutions. In contrast, perturbing only the edge costs ensures that the feasible solutions for the original problem are the same as the feasible solutions for the perturbed problem.

There is a gap between our smoothed lower bound of $\Omega(m\log(\phi))$ (Section 5.2.1) for the number of iterations that the MMCC algorithm requires and our smoothed upper bound of $O\big(mn(n\log(n) + \log(\phi))\big)$. Since our lower bound for the MMCC algorithm is weaker than the lower bound for the SSP algorithm, while the MMCC algorithm performs worse on practical instances than the SSP algorithm, we believe that our lower bound for the MMCC algorithm can be strengthened. Our stronger lower bound of $\Omega(mn)$ in case $\phi = \Omega(n^2)$ (Section 5.2.2) is another indication that this is likely possible.

Bibliography

[1] Ravindra K. Ahuja, Thomas L. Magnanti, and James B. Orlin. *Network Flows: Theory, Algorithms, and Applications.* Prentice-Hall, 1993.

[2] David Arthur, Bodo Manthey, and Heiko Röglin. Smoothed analysis of the k-means method. *Journal of the ACM*, 58(5), 2011.

[3] Mohsen Bayati, Christian Borgs, Jennifer Chayes, and Riccardo Zecchina. Belief-propagation for weighted b-matching on arbitrary graphs and its relation to linear programs with integer solutions. *SIAM Journal on Discrete Mathematics*, 25(2):989–1011, 2011.

[4] Mohsen Bayati, Alfredo Braunstein, and Riccardo Zecchina. A rigorous analysis of the cavity equations for the minimum spanning tree. *Journal of Mathematical Physics*, 49(12):125206, 2008.

[5] Mohsen Bayati, Devavrat Shah, and Mayank Sharma. Max-product for maximum weight matching: Convergence, correctness, and LP duality. *IEEE Transactions on Information Theory*, 54(3):1241–1251, 2008.

[6] René Beier and Berthold Vöcking. Random knapsack in expected polynomial time. *Journal of Computer and System Sciences*, 69(3):306–329, 2004.

[7] René Beier and Berthold Vöcking. Random knapsack in expected polynomial time. *Journal of Computer and System Sciences*, 69(3):306–329, 2004.

[8] René Beier and Berthold Vöcking. Typical properties of winners and losers in discrete optimization. *SIAM Journal in Computing*, 35(4):855–881, 2006.

[9] Robert G. Bland, Donald Goldfarb, and Michael J. Todd. The ellipsoid method: A survey. *Operations Research*, 29(6):1039–1091, November 1981.

[10] Tobias Brunsch. *Smoothed analysis of selected optimization problems and algorithms.* PhD thesis, Rheinischen Friedrich-Wilhelms-Universität Bonn, 2014.

[11] Tobias Brunsch, Kamiel Cornelissen, Bodo Manthey, and Heiko Röglin. Smoothed analysis of belief propagation for minimum-cost flow and matching. *Journal of Graph Algorithms and Applications*, 17(6):647–670, 2013. Preliminary version presented at the *7th International Workshop on Algorithms and Computation (WALCOM 2013).*

[12] Tobias Brunsch, Kamiel Cornelissen, Bodo Manthey, Heiko Röglin, and Clemens Rösner. Smoothed analysis of the successive shortest path algorithm. *SIAM Journal on Computing*, 44(6):1798–1819, 2015. Preliminary version presented at the *24th ACM-SIAM Symposium on Discrete Algorithms (SODA 2013)*.

[13] Tobias Brunsch and Heiko Röglin. Improved smoothed analysis of multiobjective optimization. In *Proc. of the 44th Ann. ACM Symp. on Theory of Computing (STOC)*, pages 681–690. ACM, 2012.

[14] Tobias Brunsch and Heiko Röglin. Finding short paths on polytopes by the shadow vertex algorithm. In Fedor V. Fomin, Rusins Freivalds, Marta Z. Kwiatkowska, and David Peleg, editors, *Automata, Languages, and Programming - 40th International Colloquium, ICALP 2013, Riga, Latvia, July 8-12, 2013, Proceedings, Part I*, volume 7965 of *Lecture Notes in Computer Science*, pages 279–290. Springer, 2013.

[15] Robert G. Busacker and Paul J. Gowen. A procedure for determining a family of miminum-cost network flow patterns. Technical Report Technical Paper 15, Operations Research Office, 1960.

[16] Amin Coja-Oghlan, Elchanan Mossel, and Dan Vilenchik. A spectral approach to analysing belief propagation for 3-colouring. *Combinatorics, Probability and Computing*, 18(6):881–912, 2009.

[17] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms (3. ed.)*. MIT Press, 2009.

[18] Kamiel Cornelissen and Bodo Manthey. Belief propagation for the maximum-weight independent set and minimum spanning tree problems. Submitted, 2015.

[19] Kamiel Cornelissen and Bodo Manthey. Smoothed analysis of the minimum-mean cycle canceling algorithm and the network simplex algorithm. In Dachuan Xu, Donglei Du, and Dingzhu Du, editors, *Proceedings of the 21st International Computing and Combinatorics Conference (COCOON 2015)*, volume 9198 of *Lecture Notes in Computer Science*, pages 701–712. Springer, 2015. Invited to appear in *Algorithmica*. Full version available at http://arxiv.org/abs/1504.08251.

[20] Daniel Dadush and Nicolai Hähnle. On the shadow simplex method for curved polyhedra. In Lars Arge and János Pach, editors, *31st International Symposium on Computational Geometry, SoCG 2015, June 22-25, 2015, Eindhoven, The Netherlands*, volume 34 of *LIPIcs*, pages 345–359. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2015.

[21] Valentina Damerow, Bodo Manthey, Friedhelm Meyer auf der Heide, Harald Räcke, Christian Scheideler, Christian Sohler, and Till Tantau. Smoothed analysis of left-to-right maxima with applications. *ACM Transactions on Algorithms*, 8(3), 2012.

[22] G. B. Dantzig. Maximization of a linear function of variables subject to linear inequalities. In T. C. Koopmans, editor, *Activity Analysis of Production and Allocation*, pages 339–347. Wiley, New York, 1951.

[23] Jack Edmonds and Richard M. Karp. Theoretical improvements in algorithmic efficiency for network flow problems. *Journal of the ACM*, 19(2):248–264, 1972.

[24] Matthias Englert, Heiko Röglin, and Berthold Vöcking. Worst case and probabilistic analysis of the 2-Opt algorithm for the TSP. In *Proc. of the 18th Ann. ACM-SIAM Symp. on Discrete Algorithms (SODA)*, pages 1295–1304. SIAM, 2007.

[25] Guy Even and Nissim Halabi. Analysis of the min-sum algorithm for packing and covering problems via linear programming. *IEEE Transactions on Information Theory*, 61(10):5295–5305, 2015.

[26] Lester R. Ford, Jr. and Delbert R. Fulkerson. *Flows in Networks.* Princeton University Press, 1962.

[27] Delbert R. Fulkerson. An out-of-kilter method for minimal cost flow problems. *Journal of the SIAM*, 9(1):18–27, 1961.

[28] David Gamarnik, Devavrat Shah, and Yehua Wei. Belief propagation for min-cost network flow: Convergence and correctness. *Operations Research*, 60(2):410–428, 2012.

[29] Andrew V. Goldberg and Robert E. Tarjan. Finding minimum-cost circulations by canceling negative cycles. *Journal of the ACM*, 36(4):873–886, 1989.

[30] Andrew V. Goldberg and Robert E. Tarjan. Finding minimum-cost circulations by successive approximation. *Mathematics of Operations Research*, 15(3):430–466, 1990.

[31] Masao Iri. A new method for solving transportation-network problems. *Journal of the Operations Research Society of Japan*, 3(1,2):27–87, 1960.

[32] William S. Jewell. Optimal flow through networks. *Operations Research*, 10(4):476–499, 1962.

[33] Richard M. Karp. A characterization of the minimum cycle mean in a digraph. *Discrete Mathematics*, 23(3):309 – 311, 1978.

[34] V. Klee and G. J. Minty. How good is the simplex algorithm? In O. Shisha, editor, *Inequalities – III*, pages 159–175. Academic Press, 1969.

[35] Morton Klein. A primal method for minimal cost flows with applications to the assignment and transportation problems. *Management Science*, 14(3):205–220, 1967.

[36] Bernhard Korte and Jens Vygen. *Combinatorial Optimization: Theory and Algorithms*. Springer, 4th edition, 2007.

[37] Péter Kovács. Minimum-cost flow algorithms: An experimental evaluation. *Optimization Methods Software*, 30(1):94–127, January 2015.

[38] Bodo Manthey. Smoothed analysis of local search algorithms. In Frank Dehne, Jörg-Rüdiger Sack, and Ulrike Stege, editors, *Proc. 14th Algorithms and Data Structures Symposium (WADS)*, volume 9214 of *Lecture Notes in Computer Science*, pages 518–527. Springer, 2015.

[39] Bodo Manthey and Heiko Röglin. Smoothed analysis: Analysis of algorithms beyond worst case. *it – Information Technology*, 53(6):280–286, 2011.

[40] George J. Minty. Monotone networks. In *Proceedings of the Royal Society of London A*, pages 194–212, 1960.

[41] Joris M. Mooij. *Understanding and Improving Belief Propagation*. PhD thesis, Radboud University Nijmegen, May 2008.

[42] James B. Orlin. Genuinely polynomial simplex and non-simplex algorithms for the minimum cost flow problem. Technical report, Sloan School of Management, MIT, Cambridge, MA, 1984. Technical Report No. 1615-84.

[43] James B. Orlin. A faster strongly polynomial minimum cost flow algorithm. *Operations Research*, 41(2):338–350, 1993.

[44] James B. Orlin. A polynomial time primal network simplex algorithm for minimum cost flows. *Mathematical Programming*, 77:109–129, 1997.

[45] Judea Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, 1988.

[46] Tomasz Radzik and Andrew V. Goldberg. Tight bounds on the number of minimum-mean cycle cancellations and related results. *Algorithmica*, 11(3):226–242, 1994.

[47] Heiko Röglin and Shang-Hua Teng. Smoothed analysis of multiobjective optimization. In *Proc. of the 50th Ann. IEEE Symp. on Foundations of Computer Science (FOCS)*, pages 681–690. IEEE, 2009.

[48] Heiko Röglin and Berthold Vöcking. Smoothed analysis of integer programming. *Mathematical Programming*, 110(1):21–56, 2007.

[49] Clemens Rösner. Smoothed analysis of the SSP algorithm and local search. Master's thesis, Rheinischen Friedrich-Wilhelms-Universität Bonn, 2014.

[50] Justin Salez and Devavrat Shah. Belief propagation: An asymptotically optimal algorithm for the random assignment problem. *Mathematics of Operation Research*, 34(2):468–480, 2009.

[51] Sujay Sanghavi, Dmitry M. Malioutov, and Alan S. Willsky. Belief propagation and LP relaxation for weighted matching in general graphs. *IEEE Transactions on Information Theory*, 57(4):2203–2212, 2011.

[52] Sujay Sanghavi and Devavrat Shah. Tightness of LP via max-product belief propagation. Technical Report 0508097v2 [cs.DS], arXiv, 2008.

[53] Sujay Sanghavi, Devavrat Shah, and Alan S. Willsky. Message passing for maximum weight independent set. *IEEE Transactions on Information Theory*, 55(11):4822 –4834, 2009.

[54] Alexander Schrijver. *Combinatorial Optimization: Polyhedra and Efficiency*. Algorithms and Combinatorics. Springer, 2003.

[55] Solomon Eyal Shimony. Finding MAPs for belief networks is NP-hard. *Artificial Intelligence*, 68(2):399–410, 1994.

[56] Daniel A. Spielman and Shang-Hua Teng. Smoothed analysis of algorithms: Why the simplex algorithm usually takes polynomial time. *Journal of the ACM*, 51(3):385–463, 2004.

[57] Daniel A. Spielman and Shang-Hua Teng. Smoothed analysis: An attempt to explain the behavior of algorithms in practice. *Communications of the ACM*, 52(10):76–84, 2009.

[58] Marshall F. Tappen and William T. Freeman. Comparison of graph cuts with belief propagation for stereo, using identical MRF parameters. In *Proc. of the 9th IEEE International Conference on Computer Vision (ICCV 2003)*, pages 900–907. IEEE Computer Society, 2003.

[59] Éva Tardos. A strongly polynomial minimum cost circulation algorithm. *Combinatorica*, 5(3):247–256, 1985.

[60] Michael J. Todd. The many facets of linear programming. *Mathematical Programming*, 91(3):417–436, February 2002.

[61] Roman Vershynin. Beyond hirsch conjecture: Walks on random polytopes and smoothed complexity of the simplex method. *SIAM Journal on Computing*, 39(2):646–678, 2009.

[62] Jens Vygen. On dual minimum cost flow algorithms. *Mathematical Methods of Operations Research*, 56(1):101–126, 2002.

[63] Yair Weiss. Correctness of local probability propagation in graphical models with loops. *Neural Computation*, 12(1):1–41, 2000.

[64] Chen Yanover and Yair Weiss. Approximate inference and protein-folding. In Suzanna Becker, Sebastian Thrun, and Klaus Obermayer, editors, *Advances in Neural Information Processing Systems (NIPS 2002)*, pages 84–86. MIT Press, 2002.

[65] Jonathan S. Yedidia, William T. Freeman, and Yair Weiss. Understanding belief propagation and its generalizations. In Gerhard Lakemeyer and Bernhard Nebel, editors, *Exploring Artificial Intelligence in the New Millennium*, chapter 8, pages 239–269. Morgan Kaufmann, 2003.

[66] Norman Zadeh. A bad network problem for the simplex method and other minimum cost flow algorithms. *Mathematical Programming*, 5(1):255–266, 1973.

Acronyms

BP	Belief propagation
BP-MST	BP algorithm for the MST problem by Bayati et al. [4]
BP-MWIS	BP algorithm for the MWIS problem by Sanghavi et al. [53]
CT-root	Root of a computation tree
IP-MWIS	Integer program for the MWIS problem
LP	Linear Program(ming)
LP-MWIS	LP relaxation of the integer program for the MWIS problem
MAP	Maximum a posteriori probability
MCF	Minimum-cost flow
MMCC algorithm	Minimum-mean cycle canceling algorithm
MRF	Markov random field
MST	Minimum(-weight) spanning tree
MST-root	Root of an MST represented as a directed tree
MWIS	Maximum-weight independent set
MWM	Maximum-weight matching
MWOST	Minimum-weight oriented spanning tree
NS algorithm	Network simplex algorithm
OST	Oriented spanning tree
SSP algorithm	Successive shortest path algorithm

Samenvatting

De traditioneel meest gebruikte manier om de prestaties van een algoritme te analyseren is *worstcase-analyse.* Hierbij worden de prestaties van het algoritme geanalyseerd in het geval dat de invoer van het algoritme bestaat uit de meest ongunstige instantie van het probleem, een zogenaamde *worstcase-instantie.* Echter, algoritmen die in theorie goed presteren voor worstcase-instanties, zijn niet altijd de algoritmen die ook in de praktijk het beste presteren. Dit komt doordat worstcase-instanties vaak erg kunstmatig zijn en in de praktijk niet of nauwelijks voorkomen. Hierdoor geeft worstcase-analyse vaak een te pessimistisch beeld van de prestaties van een algoritme in de praktijk.

Smoothed analysis is een manier om algoritmen te analyseren die vaak veel beter aansluit bij de praktijkprestaties van deze algoritmen dan worstcase-analyse. Smoothed analysis werkt als volgt. Voordat de prestaties van een algoritme voor een bepaalde instantie worden geanalyseerd, wordt deze instantie eerst willekeurig veranderd door een kleine hoeveelheid ruis toe te voegen aan de instantie. Vervolgens worden de verwachte prestaties van het algoritme na het toevoegen van de ruis geanalyseerd. Het toevoegen van een kleine hoeveelheid ruis aan worstcase-instanties zorgt er vaak voor dat de prestaties van het algoritme dramatisch verbeteren. Worstcase-instanties zijn namelijk vaak erg fragiel. De reden dat smoothed analysis goed aansluit bij prestaties van een algoritme in de praktijk is dat instanties van een probleem die zijn verkregen uit de praktijk vaak ook onderhevig zijn aan een kleine hoeveelheid ruis. Deze ruis kan bijvoorbeeld bestaan uit onnauwkeurigheid van de meetapparatuur of de precisie van de computer die wordt gebruikt om de gegevens op te slaan.

In dit proefschrift passen we smoothed analysis toe op twee klassen van algoritmen: *minimum-cost flow* algoritmen en *belief propagation* algoritmen. Het minimum-cost flow probleem is het probleem om een voorgeschreven hoeveelheid goederen door een netwerk te sturen op een zo goedkoop mogelijke manier. Het minimum-cost flow probleem is erg bekend en gedurende de afgelopen halve eeuw zijn er vele algoritmen ontwikkeld die het probleem oplossen. In dit proefschrift gebruiken we smoothed analysis om drie van deze algoritmen (het *successive shortest path* algoritme, het *minimum-mean cycle canceling* algoritme en het *network simplex* algoritme) te analyseren en bewijzen we onder- en bovengrenzen voor de looptijd van deze algoritmen.

Het *belief propagation* (BP) algoritme (in dit proefschrift analyseren we de zogenaamde *max-product* variant van belief propagation) is een algoritme voor het oplossen van *probabilistische inferentieproblemen.* Probabilistische inferentieproble-

men bestaan uit het afleiden van bepaalde eigenschappen van kansverdelingen. Voorbeelden zijn het bepalen van de marginale verdeling van een stochastische variabele of het bepalen van een *maximum-a-posteriori-schatter* (de meest waarschijnlijke simultane realisatie van een collectie stochastische variabelen) van een kansverdeling. Het BP algoritme is een zogenaamd *message passing* algoritme. Elke stochastische variabele probeert zijn optimale waarde te bepalen door te communiceren met zijn buurvariabelen door middel van het uitwisselen van berichten. Het BP algoritme is erg populair aangezien het eenvoudig is en vaak goed presteert in de praktijk. Er is echter nog weinig bekend over het theoretische gedrag van het BP algoritme. Om de werking van het BP algoritme beter te begrijpen vanuit theoretisch oogpunt, passen we het toe op enkele bekende optimalisatieproblemen. In dit proefschift onderzoeken we onder welke condities het BP algoritme convergeert naar de correcte oplossing en passen we smoothed analysis toe om de looptijd van het BP algoritme te analyseren.

About the Author

Kamiel Cornelissen was born in Utrecht, the Netherlands, on December 14, 1980. Soon after, he moved to Nijmegen, where he received his gymnasium diploma (secondary education) from the NSG in 1999. Afterwards, he moved to Enschede, where he obtained a master's degree (cum laude) in Applied Mathematics at the University of Twente in 2009. During his studies, he performed an internship at Paragon Decision Technology, where he developed a game to introduce new users to the AIMMS software. His final project "Algorithmic feature generation for microscale topographies" was a cooperation of the Tissue Regeneration group and the Discrete Mathematics and Mathematical Programming (DMMP) group of the University of Twente. After obtaining his master's degree, he continued the research done for his final project as an academy assistant for the Royal Netherlands Academy of Arts and Sciences (KNAW).

In 2011 Kamiel Cornelissen started his Ph.D. research under the supervision of dr. Bodo Manthey and prof. dr. Marc Uetz in the DMMP group at the University of Twente. The topics of his research were smoothed analysis, belief propagation, and minimum-cost flow algorithms. His Ph.D. research culminates with this thesis and the defense on May 27, 2016.

While performing his master's and Ph.D. work, Kamiel Cornelissen competed in the professional tournament circuit of the strategy card game Magic: The Gathering. In 2009 he was inducted in the Magic: The Gathering Pro Tour Hall of Fame as a recognition of his performance.

Algorithms that have good worst-case performance are not always the ones that perform best in practice. The smoothed analysis framework is a way of analyzing algorithms that usually matches practical performance of these algorithms much better than worst-case analysis.

In this thesis we apply smoothed analysis to two classes of algorithms: minimum-cost flow algorithms and belief propagation algorithms. The minimum-cost flow problem is the problem of sending a prescribed amount of flow through a network in the cheapest possible way. It is very well known, and over the last half a century many algorithms have been developed to solve it. We analyze three of these algorithms (the successive shortest path algorithm, the minimum-mean cycle canceling algorithm, and the network simplex algorithm) in the framework of smoothed analysis and show lower and upper bounds on their smoothed running-times.

The belief propagation algorithm is a message-passing algorithm for solving probabilistic inference problems. Because of its simplicity, it is very popular in practice. However, its theoretical behavior is not well understood. To obtain a better theoretical understanding of the belief propagation algorithm, we apply it to several well-studied optimization problems. We analyze under which conditions the belief propagation algorithm converges to the correct solution and we analyze its smoothed running-time.

CTIT

UNIVERSITY OF TWENTE.

CTIT Ph.D. Thesis Series No. 16-385 ISSN: 1381-3617 ISBN: 978-90-365-4097-1