Smoothed Analysis of the Successive Shortest Path Algorithm[*]

Tobias Brunsch [†] Kamiel Cornelissen [‡] Bodo Manthey [§] Heiko Röglin [¶]

Abstract

The minimum-cost flow problem is a classic problem in combinatorial optimization with various applications. Several pseudo-polynomial, polynomial, and strongly polynomial algorithms have been developed in the past decades, and it seems that both the problem and the algorithms are well understood. However, some of the algorithms' running times observed in empirical studies contrast the running times obtained by worst-case analysis not only in the order of magnitude but also in the ranking when compared to each other. For example, the Successive Shortest Path (SSP) algorithm, which has an exponential worst-case running time, seems to outperform the strongly polynomial Minimum-Mean Cycle Canceling algorithm. To explain this discrepancy, we study the SSP algorithm in the framework of smoothed analysis and establish a bound of $O(mn\phi(m + n \log n))$ for its smoothed running time. This shows that worst-case instances for the SSP algorithm are not robust and unlikely to be encountered in practice.

1 Introduction

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, 7]. Plenty of algorithms have been developed over the last fifty years. The first pseudo-polynomial algorithm for the minimum-cost flow problem was the Out-of-Kilter algorithm independently proposed by Minty [17] and by Fulkerson [8]. The simplest pseudo-polynomial algorithms are the primal Cycle Canceling algorithm by Klein [14] and the dual Successive Shortest Path (SSP) algorithm by Jewell [12], Iri [11], and Busacker and Gowen [4]. By introducing a scaling technique Edmonds and Karp [6] modified the SSP algorithm to obtain the Capacity Scaling algorithm, which was the first polynomial time algorithm for the minimum-cost flow problem.

The first strongly polynomial algorithms were given by Tardos [23] and by Orlin [18]. Later, Goldberg and Tarjan [9] 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 [19] and has a running time of $O(m \log(n)(m+n \log n))$. For an extensive overview of minimum-cost flow algorithms we suggest the paper of Goldberg and Tarjan [10], the paper of Vygen [25], and the book of Ahuja, Magnanti, and Orlin [1].

Zadeh [26] showed that the SSP algorithm has an exponential worst-case running time. Contrary to this, the worst-case running times of the Capacity Scaling algorithm and the MMCC algorithm are $O(m(\log U)(m+n \log n))$ [6] and $O(m^2n^2 \min\{\log(nC), m\})$ [20], respectively. Here, U denotes the maximum edge capacity and C denotes the maximum edge cost. In particular, the former is polynomial whereas the latter is even strongly polynomial. However, the notions of pseudo-polynomial, polynomial, and strongly polynomial algorithms always refer to worst-case running times, which do not always resemble the algorithms' behavior on real-life instances. Algorithms with large worst-case running times do not inevitably perform poorly in practice. An experimental study of Király and Kovács [13] indeed observes running time behaviors significantly deviating from what the worst-case running times indicate. The MMCC algorithm is completely outperformed by the SSP algorithm. The Capacity Scaling algorithm is the fastest of these three algorithms, but its running time seems to be in the same order of magnitude as the running time of the SSP algorithm. In this paper, we explain why the SSP algorithm comes off so well by applying the framework of smoothed analysis.

Smoothed analysis was introduced by Spielman and Teng [21] to explain why the simplex method is efficient in practice despite its exponential worst-case running time. In the original model, an adversary chooses an arbitrary instance which is subsequently slightly perturbed at random. In this way, pathological instances no longer dominate the analysis. Good smoothed bounds usually indicate good behavior in practice be-

[*]This research was supported by ERC Starting Grant 306465 (BeyondWorstCase) and NWO grant 613.001.023.

[†]University of Bonn, Department of Computer Science, Germany. Email: `brunsch@cs.uni-bonn.de`

[‡]University of Twente, Department of Applied Mathematics, Enschede, The Netherlands. Email: `k.cornelissen@utwente.nl`

[§]University of Twente, Department of Applied Mathematics, Enschede, The Netherlands. Email: `b.manthey@utwente.nl`

[¶]University of Bonn, Department of Computer Science, Germany. Email: `heiko@roeglin.org`

cause in practice inputs are often subject to a small amount of random noise. For instance, this random noise can stem from 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. Since its invention, smoothed analysis has been successfully applied in a variety of contexts. Two recent surveys [16, 22] summarize some of these results.

We follow a more general model of smoothed analysis due to Beier and Vöcking [2]. In this model, the adversary is even allowed to specify the probability distribution of the random noise. The power of the adversary is only limited by the *smoothing parameter* ϕ. In particular, in our input model the adversary does not fix the edge costs $c_e \in [0,1]$ for each edge e, but he specifies probability density functions $f_e \colon [0,1] \to [0,\phi]$ according to which the costs c_e are randomly drawn independently of each other. 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 small intervals I_e of length $1/\phi$ (that contain the worst-case costs) for each edge e from which the costs c_e are drawn uniformly.

As in the worst-case analysis, the network graph, the edge capacities, and the balance values of the nodes are chosen adversarially. The edge capacities and the balance values of the nodes are even allowed to be real values. We define the smoothed running time of an algorithm as the worst expected running time the adversary can achieve and we prove the following theorem.

Theorem 1.1. *The smoothed running time of the SSP algorithm is $O(mn\phi(m + n\log n))$.*

If ϕ is a constant – which seems to be a reasonable assumption if it models, for example, measurement errors – then the smoothed bound simplifies to $O(mn(m + n\log n))$. Hence, it is unlikely to encounter instances on which the SSP algorithm requires an exponential amount of time. Still, this bound is worse than the bound $O(m\log(n)(m + n\log n))$ of Orlin's Enhanced Capacity Scaling algorithm, but this coincides with practical observations.

Now let us compare the running times of the SSP algorithm and the strongly polynomial MMCC algorithm. Radzik and Goldberg [20] presented a family of worst-case instances on which the MMCC algorithm requires $\Omega(m^2 n)$ minimum-mean cost cycle cancellations, which for $m = \Omega(n\log n)$ is already in the same order as the smoothed running time of the SSP algorithm. Together with the observation that the best known algorithm for computing minimum-mean cost cycles has a running time of $\Theta(mn)$ [5], this can be viewed as the first theoretical indication for why the SSP algorithm clearly outperforms the strongly polynomial MMCC algorithm in experiments. However, admittedly this comparison is not entirely fair because it compares the smoothed running time of the SSP algorithm with the worst-case running time of the MMCC algorithm. For a complete explanation why the SSP algorithm outperforms the MMCC algorithm in practice a smoothed lower bound of the running time of the latter one must be derived that exceeds the upper bound stated in Theorem 1.1.

Although the MMCC algorithm was mainly considered for theoretical reasons and not necessarily intended for practical purposes, this comparison serves as an example for a phenomenon which can be explained by smoothed analysis but not by worst-case analysis.

The main technical section of this paper is devoted to the proof of Theorem 1.1 (Section 4). At the end of this paper (Section 5), we point out some connections between SSP and its smoothed analysis to the simplex method with the shadow vertex pivot rule, which has been used by Spielman and Teng in their smoothed analysis [21].

1.1 The Minimum-Cost Flow Problem

A *flow network* is a simple directed graph $G = (V,E)$ together with a *capacity function* $u \colon E \to \mathbb{R}_{\geq 0}$. For convenience, we assume that there are no directed cycles of length two. In the minimum-cost flow problem there are an additional *cost function* $c \colon E \to [0,1]$ and a *balance 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 $f \colon E \to \mathbb{R}_{\geq 0}$ that obeys the capacity constraints $0 \leq f(e) \leq u(e)$ for any edge $e \in E$ and Kirchhoff's law adapted to the balance values, i.e., $b(v) + \sum_{e=(u,v)\in E} f(e) = \sum_{e'=(v,w)\in E} f(e')$ for all nodes $v \in V$. The cost of a feasible b-flow is 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.

1.2 The SSP Algorithm

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 vertex set V, arc set $E' = E_\mathrm{f} \cup E_\mathrm{b}$, where

$$E_\mathrm{f} = \big\{ e :\ e \in E \text{ and } f(e) < u(e) \big\}$$

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

$$E_\mathrm{b} = \left\{e^{-1} : e \in E \text{ and } f(e) > 0\right\}$$

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

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

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 balance value) and one *demand node* (a node with negative balance value). 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 maximum of the sum of the capacities of the auxiliary edges incident with s and the sum of the capacities of the auxiliary edges incident with t. All other balance values are set to 0. The SSP algorithm run on this instance computes the minimum-cost b-flow for the original instance. In the remainder of this paper 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., $|f| = \sum_{e=(s,v)\in E} f(e)$.

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.

Theorem 1.2. *In any round i, flow f_i is a minimum-cost b_i-flow for the balance function b_i defined by $b_i(s) = -b_i(t) = |f_i|$ and $b_i(v) = 0$ for $v \notin \{s, t\}$.*

Theorem 1.2 is due to Jewell [12], Iri [11], and Busacker and Gowen [4]. We refer to Korte and Vygen [15] 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.

1.3 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. Let us first consider the case that all edge costs are integers from $\{1, \ldots, C\}$. In this case the length of any possible path 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 $O(mn)$. Hence, the SSP algorithm terminates within $O(mn^2C)$ 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. Indeed, instances can be generated for which the number of augmentation steps is exponential in m and n. Nevertheless, an immediate consequence of Theorem 1.1 is that, in expectation, the SSP algorithm terminates within $O(mnC)$ steps on instances of this form.

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.)

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. To distinguish between the original network G and some residual network G_f in the remainder of this paper, 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

Algorithm 1 SSP for single-source-single-sink minimum-cost flow networks with $b(s) = -b(t) = z$.

```
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 output an error
4:     find a shortest $s$-$t$-path $P_i$ in $G_{f_{i-1}}$ with respect to the arc costs
5:     augment the flow as much as possible* along path $P_i$ to obtain a new flow $f_i$
6:     if $|f_i| = z$ then output $f_i$
7: end for
```

*The value $|f_i|$ of flow f_i must not exceed z and f_i must obey all capacity constraints.

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. With $u \overset{P}{\rightsquigarrow} v$, we refer to the subpath of P starting from node u going to node v. 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 in possible residual networks are called *possible paths*.

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 $L = (\ell_0, \ell_1, \dots)$ of real numbers. We show that this sequence is strictly monotonically increasing with a probability of 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 subintervals 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. Thus, it suffices to bound the probability that an element of L falls into some interval $(d, d + \varepsilon]$.

For this, assume 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 flow f_{i-1} without knowing the cost of edge e_0 that corresponds to arc e in the original network. Hence, we do not have to reveal c_{e_0} for this. However, the length of P_i, which equals ℓ_i, depends linearly on c_{e_0}, and the coefficient is $+1$ or -1. 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 [3], 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 Analysis of the SSP Algorithm

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.

LEMMA 4.1. *Let $d_i(v)$ denote the distance from s to node v in the residual network G_{f_i}. Then the sequence $d_0(v), d_1(v), d_2(v), \dots$ is monotonically increasing.*

Proof. 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 4.1. *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 reals x from the interval $[0, |f_{\max}|]$ to the cost of the cheapest flow f with value x, i.e., $\mathscr{C}(v) = \min\{c(f) : |f| = x\}$.*

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, we will reveal a connection between the values $\ell_-(f_i)$ and the function $\mathscr{C}$. Based on this, we can focus on analyzing function $\mathscr{C}$.

COROLLARY 4.1. *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. *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 1.2 and the observation that the cost of the flow is 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 4.1. *Consider the flow network depicted in Figure 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, Table 1 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 2).*

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)

Table 1: The augmenting paths for Example 4.1.

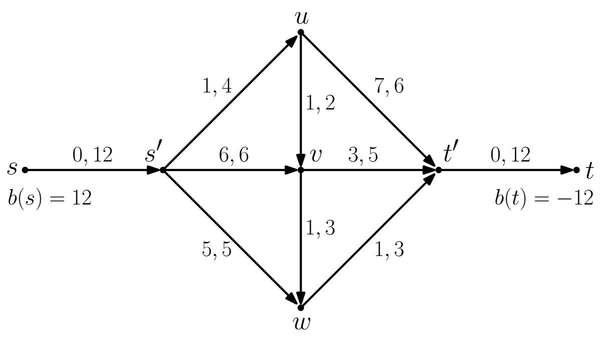

Figure 1: Minimum-cost flow network with master source s and master sink t.

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. *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.

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 4.3. *Let $f \in \mathcal{F}_0$ be a predecessor of a good flow for which $\ell_-^G(f) < \ell_+^G(f)$ holds, and let $d \in [\ell_-^G(f), \ell_+^G(f))$ be an arbitrary real number. 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 $\ell_-^{G'}(f) \leq \ell_-^G(f) \leq d < \ell_+^G(f) \leq \ell_+^{G'}(f)$ 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. In particular, they are continuous and the breakpoints correspond to the values of the flows $\tilde{f} \in \mathcal{F}_0(G)$ and

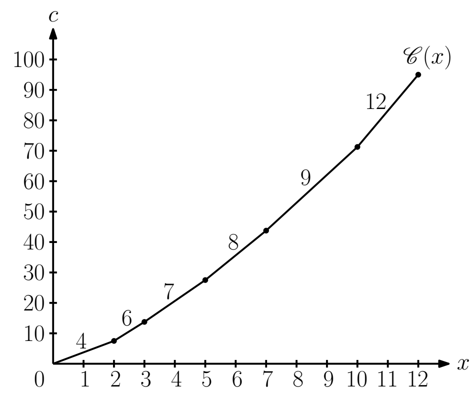

Figure 2: Cost function $\mathscr{C}$.

$\hat{f} \in \mathcal{F}_0(G')$ with $\ell_-^G(\tilde{f}) < \ell_+^G(\tilde{f})$ and $\ell_-^{G'}(\hat{f}) < \ell_+^{G'}(\hat{f})$, respectively.

We start with analyzing the case $e_0 = e$. In this case, we set $\mathscr{C}'' = \mathscr{C}'$ and observe that by increasing the cost of edge e_0 to 1 the cost of no flow can decrease. Hence, $\mathscr{C}'' \geq \mathscr{C}$. 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 cost of each flow is 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, $\mathscr{C}'' \geq \mathscr{C}$ and $\mathscr{C}''(|f|) = \mathscr{C}(|f|)$.

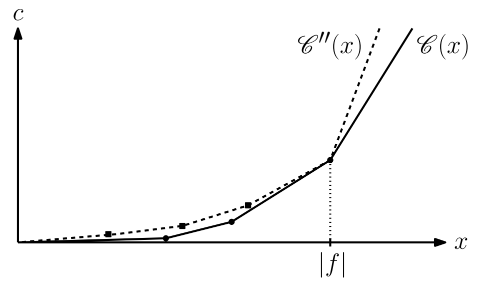

Figure 3: Cost function $\mathscr{C}$ and function $\mathscr{C}''$.

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 3). Since $|f|$ is a breakpoint of $\mathscr{C}$, this implies that $|f|$ is also a breakpoint of $\mathscr{C}''$ and that the slope to the left of $\mathscr{C}''$ at $|f|$ is at most the slope to the left of $\mathscr{C}$ at $|f|$. 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 to the right of $\mathscr{C}''$ at $|f|$ is at least the slope to the right of $\mathscr{C}$ at $|f|$. These properties carry over to $\mathscr{C}'$. Hence, $f \in \mathcal{F}_0(G')$. Recalling $d \in \big[\ell_-^G(f), \ell_+^G(f)\big)$ and the fact that the slopes correspond to shortest s-t-path lengths, the stated chain of inequalities follows. □

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 $f'$
```

Lemma 4.3 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. *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. Then* Reconstruct(e, d) *outputs flow f.*

Proof. By applying Lemma 4.3, we obtain $f \in \mathcal{F}_0(G')$ and $\ell_-^{G'}(f) \leq d < \ell_+^{G'}(f)$. Together with Corollary 4.1, 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 is an essential component of the proof of Theorem 1.1 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 4.4. *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. □

According to Lemma 4.4 we can assume that there is no s-t-path of length 0 and that the following property holds since it holds with a probability of 1.

Property 4.1. *For any nodes u and v the lengths of all possible u-v-paths are pairwise distinct.*

Lemma 4.5. *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 through the augmentation along P_i. Let v be the first node of P that occurs in the subpath $y \overset{P_i}{\rightsquigarrow} t$ of P_i, and let u be the last node in the subpath $s \overset{P}{\rightsquigarrow} v$ of P that belongs to the subpath $s \overset{P_i}{\rightsquigarrow} x$ of P_i (see Figure 4). By the choice of u and v, all nodes on the subpath $P' = u \overset{P}{\rightsquigarrow} v$ 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}.

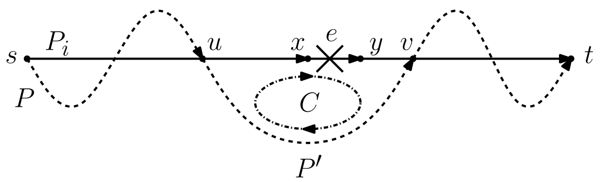

Figure 4: Paths P and P_i in the residual network G_{f_i}.

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 $u \overset{P_i}{\rightsquigarrow} v$ and the reverse arcs e^{-1} of 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 subintervals of length ε and treat the number of lengths $\ell_{-}(f_i)$ that fall into a given subinterval 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 4.4) we will simply bound the number of augmentation steps of the SSP algorithm by a worst-case bound according to the following lemma.

Lemma 4.6. *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. Hence, for two distinct flows $f_i, f_j \in \mathcal{F}_0$, the residual networks G_{f_i} and G_{f_j} are not equivalent due to Corollary 4.1 and Property 4.1. 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 4.7. *There are at most n bad flows $f \in \mathcal{F}$.*

Proof. According to Lemma 4.5, 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 subinterval 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 subinterval of length ε.

Lemma 4.8. *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 $\mathbf{Pr}[\mathsf{E}_{d,\varepsilon}] \leq 2m\varepsilon\phi + 2 \cdot \mathbf{Pr}[B_{d,\varepsilon}]$.*

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 $\mathbf{Pr}[A_{d,\varepsilon}] \leq 2m\varepsilon\phi + \mathbf{Pr}[B_{d,\varepsilon}]$. 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 4.1 and Property 4.1 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 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: By calling $\mathsf{Reconstruct}(e, d)$, we obtain a certain flow f. Let ℓ be the length of the shortest s-t-path in G_f that uses arc e. Then event $R_{e,d,\varepsilon}$ occurs if $\ell \in (d, d+\varepsilon]$. Therefore, the probability of event $A_{d,\varepsilon}$ is bounded by

$$\sum_{e \in E} \mathbf{Pr}[R_{e,d,\varepsilon}] + \sum_{e \in E} \mathbf{Pr}\left[R_{e^{-1},d,\varepsilon}\right] + \mathbf{Pr}[B_{d,\varepsilon}] .$$

We conclude the proof by showing $\mathbf{Pr}[R_{e,d,\varepsilon}] \leq \varepsilon\phi$. 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, $\ell = \ell' + c_e = \ell' + \mathsf{sgn}(e) \cdot c_{e_0}$, where ℓ' is the length of P_1 plus the length of P_2 and where

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

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 4.3. *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 4.8, 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 4.1, 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,

$$\begin{aligned} \mathbf{E}[X] &\leq \sum_{i=0}^{k-1} \mathbf{E}\left[Z_{i \cdot \frac{n}{k}, \frac{n}{k}}\right] + 3^{m+n} \cdot \mathbf{Pr}\left[F_{\frac{n}{k}}\right] \\ &= \sum_{i=0}^{k-1} \mathbf{Pr}\left[\mathsf{E}_{i \cdot \frac{n}{k}, \frac{n}{k}}\right] + 3^{m+n} \cdot \mathbf{Pr}\left[F_{\frac{n}{k}}\right] \\ &\leq 2mn\phi + 2 \cdot \sum_{i=0}^{k-1} \mathbf{Pr}\left[B_{i \cdot \frac{n}{k}, \frac{n}{k}}\right] + 3^{m+n} \cdot \mathbf{Pr}\left[F_{\frac{n}{k}}\right] \\ &\leq 2mn\phi + 2n + 3^{m+n} \cdot \mathbf{Pr}\left[F_{\frac{n}{k}}\right] . \end{aligned}$$

The second inequality is due to Lemma 4.8 whereas the third inequality stems from Lemma 4.7. The claim follows since $\mathbf{Pr}\left[F_{\frac{n}{k}}\right] \to 0$ for $k \to \infty$ in accordance with Lemma 4.4. □

Now we are almost done with the proof of our main theorem.

Proof of Theorem 1.1. Since each step of the SSP algorithm runs in time $O(m+n\log n)$ using Dijkstra's algorithm (see, e.g., Korte [15] for details), applying Corollary 4.3 yields the desired result.

5 Smoothed Analysis of the Simplex Algorithm

In this section we describe a surprising connection between our result about the SSP algorithm and the smoothed analysis of the simplex algorithm. Spielman and Teng's original smoothed analysis [21] as well as Vershynin's [24] 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^* 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^*. 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 $(|f|, u(f))$. Theorem 1.2 guarantees that the cost function depicted in Figure 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 1.1 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 algorithm 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 [21] and Vershynin's [24] smoothed analysis 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.

References

[1] Ravindra K. Ahuja, Thomas L. Magnanti, and James B. Orlin. *Network flows – theory, algorithms and applications.* Prentice Hall, 1993.

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

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

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

[5] Ali Dasdan and Rajesh K. Gupta. Faster maximum and minimum mean cycle algorithms for system-performance analysis. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 17:889–899, 1997.

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

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

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

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

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

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

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

[13] Zoltán Király and Péter Kovács. Efficient implementations of minimum-cost flow algorithms. *Acta Universitatis Sapientiae, Informatica*, 4(1):67–118, 2012.

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

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

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

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

[18] 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.

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

[20] 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.

[21] 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.

[22] 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.

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

[24] 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.

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

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