Smoothed Analysis of the Minimum-Mean Cycle Canceling Algorithm and the Network Simplex Algorithm*

Kamiel Cornelissen and Bodo Manthey

University of Twente, Enschede, The Netherlands
k.cornelissen@utwente.nl, b.manthey@utwente.nl

Abstract. The minimum-cost flow (MCF) problem is a fundamental optimization problem with many applications and seems to be well understood. Over the last half century many algorithms have been developed to solve the MCF problem and these algorithms have varying worst-case bounds on their running time. However, these worst-case bounds are not always a good indication of the algorithms' performance in practice. The Network Simplex (NS) algorithm needs an exponential number of iterations for some instances, but it is considered the best algorithm in practice and performs best in experimental studies. On the other hand, the Minimum-Mean Cycle Canceling (MMCC) algorithm is strongly polynomial, but performs badly in experimental studies.
To explain these differences in performance in practice we apply the framework of smoothed analysis. For the number of iterations of the MMCC algorithm we show an upper bound of $O(mn^2 \log(n) \log(\phi))$. Here n is the number of nodes, m is the number of edges, and ϕ is a parameter limiting the degree to which the edge costs are perturbed. We also show a lower bound of $\Omega(m \log(\phi))$ for the number of iterations of the MMCC algorithm, which can be strengthened to $\Omega(mn)$ when $\phi = \Theta(n^2)$. For the number of iterations of the NS algorithm we show a smoothed lower bound of $\Omega(m \cdot \min\{n, \phi\} \cdot \phi)$.

1 Introduction

The minimum-cost flow (MCF) problem is a well-studied problem with many applications in, for example, modeling transportation and communication networks [1, 7]. Over the last half century many algorithms have been developed to solve it. The first algorithms proposed in the 1960s were all pseudo-polynomial. These include the Out-of-Kilter algorithm by Minty [17] and by Fulkerson [8], the Cycle Canceling algorithm by Klein [13], the Network Simplex (NS) algorithm by Dantzig [5], and the Successive Shortest Path (SSP) algorithm by Jewell [11], Iri [10], and Busacker and Gowen [4]. In 1972 Edmonds and Karp [6] proposed the Capacity Scaling algorithm, which was the first polynomial MCF algorithm. In the 1980s the first strongly polynomial algorithms were developed by Tardos [24] and by Orlin [18]. Later, several more strongly polynomial algorithms

* A full version with all proofs is available at http://arxiv.org/abs/1504.08251.

were proposed such as the Minimum-Mean Cycle Canceling (MMCC) algorithm by Goldberg and Tarjan [9] and the Enhanced Capacity Scaling algorithm by Orlin [19], which currently has the best worst-case running time. For a more complete overview of the history of MCF algorithms we refer to Ahuja et al. [1].

When we compare the performance of several MCF algorithms in theory and in practice, we see that the algorithms that have good worst-case bounds on their running time are not always the ones that perform best in practice. Zadeh [25] showed that there exist instances for which the Network Simplex (NS) algorithm has exponential running time, while the Minimum-Mean Cycle Canceling (MMCC) algorithm runs in strongly polynomial time, as shown by Goldberg and Tarjan [9]. In practice however, the relative performance of these algorithms is completely different. Kovács [15] showed in an experimental study that the NS algorithm is much faster than the MMCC algorithm on practical instances. In fact, the NS algorithm is even the fastest MCF algorithm of all. An explanation for the fact that the NS algorithm performs much better in practice than indicated by its worst-case running time is that its worst-case instances are very contrived and unlikely to occur in practice. To better understand the practical performance for the NS algorithm and the MMCC algorithm, we analyze these algorithms in the framework of smoothed analysis.

Smoothed analysis was introduced by Spielman and Teng [22] to explain why the simplex algorithm usually needs only a polynomial number of iterations in practice, while in the worst case it needs an exponential number of iterations. In the framework of smoothed analysis, an adversary can specify any instance and this instance is then slightly perturbed before it is used as input for the algorithm. This perturbation can model, for example, measurement errors or numerical imprecision. In addition, it can model noise on the input that can not be quantified exactly, but for which there is no reason to assume that it is adversarial. Algorithms that have a good smoothed running time often perform well in practice. We refer to two surveys [16, 23] for a summary of results that have been obtained using smoothed analysis.

We consider a slightly more general model of smoothed analysis, introduced by Beier and Vöcking [2]. In this model the adversary can not only specify the mean of the noisy parameter, but also the type of noise. We use the following smoothed input model for the MCF problem. An adversary can specify the structure of the flow network including all nodes and edges, and also the exact edge capacities and budgets of the nodes. However, the adversary can not specify the edge costs exactly. For each edge e the adversary can specify a probability density $g_e : [0, 1] \to [0, \phi]$ according to which the cost of e is drawn at random. The parameter ϕ determines the maximum density of the density function and can therefore be interpreted as the power of the adversary. If ϕ is large, the adversary can very accurately specify each edge cost and we approach worst-case analysis. If $\phi = 1$, the adversary has no choice but to specify the uniform density on the interval $[0, 1]$ and we have average-case analysis.

Brunsch et al. [3] were the first to show smoothed bounds on the running time of an MCF algorithm. They showed that the SSP algorithm needs $O(mn\phi)$

iterations in expectation and has smoothed running time $O(mn\phi(m+n\log(\phi)))$, since each iteration consists of finding a shortest path. They also provide a lower bound of $\Omega(m\cdot\min\{n,\phi\}\cdot\phi)$ for the number of iterations that the SSP algorithm needs, which is tight for $\phi=\Omega(n)$. These bounds show that the SSP algorithm needs only a polynomial number of iterations in the smoothed setting, in contrast to the exponential number it needs in the worst case, and explains why the SSP algorithm performs quite well in practice. In order to fairly compare the SSP algorithm with other MCF algorithms in the smoothed setting, we need smoothed bounds on the running times of these other algorithms. Brunsch et al. [3] asked particularly for smoothed running time bounds for the MMCC algorithm, since the MMCC algorithm has a much better worst-case running time than the SSP algorithm, but performs worse in practice. It is also interesting to have smoothed bounds for the NS algorithm, since the NS algorithm is the fastest MCF algorithm in practice. However, until now no smoothed bounds were known for other MCF algorithms. In this paper we provide smoothed lower and upper bounds for the MMCC algorithm, and a smoothed lower bound for the NS algorithm.

For the MMCC algorithm we prove an upper bound of $O(mn^2\log(n)\log(\phi))$ for the expected number of iterations that the MMCC algorithm needs (Section 2). 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).

We also provide a lower bound (Section 3.1) on the number of iterations that the MMCC algorithm needs. For every n, every $m\in\{n,n+1,\dots,n^2\}$, and every $\phi\leq 2^n$, we provide an instance with $\Theta(n)$ nodes and $\Theta(m)$ edges for which the MMCC algorithm requires $\Omega(m\log(\phi))$ iterations. For $\phi=\Omega(n^2)$ we can improve our lower bound (Section 3.2). We show that for every $n\geq 4$ and every $m\in\{n,n+1,\dots,n^2\}$, 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. This is indeed a stronger lower bound than the bound for general ϕ, since we have $m\log(\phi)=\Theta(m\log(n))$ for $\phi=\Theta(n^2)$.

For the NS algorithm we provide a lower bound (Section 4) on the number of non-degenerate iterations that it requires. In particular, we show that for every n, every $m\in\{n,\dots,n^2\}$, and every $\phi\leq 2^n$ there exists a flow network with $\Theta(n)$ nodes and $\Theta(m)$ edges, and an initial spanning tree structure for which the NS algorithm needs $\Omega(m\cdot\min\{n,\phi\}\cdot\phi)$ non-degenerate iterations with probability 1. The existence of an upper bound is our main open problem. Note that our bound is the same as the lower bound that Brunsch et al. [3] found for the smoothed number of iterations of the SSP algorithm. 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).

In the rest of our introduction we introduce the MCF problem, the MMCC algorithm and the NS algorithm in more detail. In the rest of our paper, all logarithms are base 2.

1.1 Minimum-Cost Flow Problem

A *flow network* is a simple directed graph $G = (V, E)$ together with a nonnegative capacity function $u : E \to \mathbb{R}_+$ defined on the edges. For convenience, we assume that G is connected and that E does not contain a pair (u, v) and (v, u) of reverse edges. For the MCF problem, we also have a cost function $c : E \to [0, 1]$ on the edges and a budget function $b : V \to \mathbb{R}$ on the nodes. Nodes with negative budget require a resource, while nodes with positive budget offer it. A *flow* $f : E \to \mathbb{R}_+$ is a nonnegative function on the edges that satisfies the capacity constraints, $0 \leq f(e) \leq u(e)$ (for all $e \in E$), and flow conservation constraints $b(v) + \sum_{e=(u,v)\in E} f(e) = \sum_{e'=(v,w)\in E} f(e')$ (for all $v \in V$). The cost $c(f)$ of a flow f is defined as the sum of the flow on each edge times the cost of that edge, that is, $c(f) = \sum_{e\in E} c(e) \cdot f(e)$. The objective of the minimum-cost flow problem is to find a flow of minimum cost or conclude that no feasible flow exists.

In our analysis we often use the concept of a *residual network*, which we define here. For an edge $e = (u, v)$ we denote the reverse edge (v, u) by e^{-1}. For flow network G and flow f, the residual network G_f is defined as the graph $G_f = (V, E_f \cup E_b)$. Here $E_f = \{e \mid e \in E \text{ and } f(e) < u(e)\}$ is the set of *forward edges* with capacity $u'(e) = u(e) - f(e)$ and cost $c'(e) = c(e)$. $E_b = \{e \mid e^{-1} \in E \text{ and } f(e^{-1}) > 0\}$ is the set of *backward edges* with capacity $u'(e) = f(e^{-1})$ and cost $c'(e) = -c(e^{-1})$. Here $u'(e)$ is also called the *residual capacity* of edge e for flow f.

1.2 Minimum-Mean Cycle Canceling Algorithm

The MMCC algorithm works as follows:

- First we find a feasible flow using any maximum-flow algorithm.
- Next, as long as the residual network contains cycles of negative total cost, we find a cycle of minimum-mean cost and maximally augment flow along this cycle.
- We stop when the residual network does not contain any cycles of negative total cost.

For a more elaborate description of the MMCC algorithm, we refer to Korte and Vygen [14]. In the following, we denote the mean cost of a cycle C by $\mu(C) = \left(\sum_{e\in C} c(e)\right)/|C|$. Also, for any flow f, we denote the mean cost of the cycle of minimum-mean cost in the residual network G_f by $\mu(f)$.

Goldberg and Tarjan [9] proved in 1989 that the Minimum-Mean-Cycle Canceling algorithm runs in strongly polynomial time. Five years later Radzik and Goldberg [21] 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 [12]. The tight bound on the number of iterations that the MMCC algorithm needs is as follows.

Theorem 1.1 (Radzik and Goldberg). *The number of iterations needed by the MMCC algorithm is bounded by $O(nm^2)$ and this bound is tight.*

To prove our smoothed bounds in the next sections, we use another result by Korte and Vygen [14, Corollary 9.9] which states that the absolute value of the mean cost of the cycle that is canceled by the MMCC algorithm, $|\mu(f)|$, decreases by at least a factor $1/2$ every mn iterations.

Theorem 1.2 (Korte and Vygen). *Every mn iterations of the MMCC algorithm, $|\mu(f)|$ decreases by at least a factor $1/2$.*

1.3 Network Simplex Algorithm

The Network Simplex (NS) algorithm starts with an initial spanning tree structure (T, L, U) and associated flow f, where each edge in E is assigned to exactly one of T, L, and U, and it holds that

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

If the MCF problem has a feasible solution, such a structure can always be found by first finding any feasible flow and then augmenting flow along cycles consisting of only edges that have a positive amount of flow less than their capacity, until no such cycles remain. Note that the structure (T, L, U) uniquely determines the flow f, since the edges in T form a tree. In addition to the spanning tree structure, the NS algorithm also keeps track of a set of node potentials $\pi(v)$ 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 $c^\pi(u, v) = c(u, v) - \pi(u) + \pi(v)$ of an edge (u, v) equals 0 for all edges $(u, v) \in T$.

In each iteration, the NS algorithm tries to improve the current flow by adding an edge to T that violates its optimality condition. An edge in L violates its optimality condition if it has strictly negative reduced cost, while an edge in U violates its optimality condition if it has strictly positive reduced cost. One of the edges e that violates its optimality condition is added to T, which creates a unique cycle C in T. Flow is maximally augmented along C, until the flow on one of the edges $e' \in C$ becomes 0 or reaches its capacity. The edge e' leaves T, after which T is again a spanning tree of G. Next we update the sets T, L, and

U, 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 is called the entering edge and the edge e' that leaves T is called the leaving edge. Note that in some cases the entering edge can be the same edge as the leaving edge. Also, if one of the edges in the cycle C already contains flow equal to its capacity, the flow is not changed in that iteration, but the spanning tree T 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 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 $|c^\pi(e)|$ is maximum. In case multiple edges in C are candidates to be the leaving edge, we choose the one that is most convenient for our analysis.

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 [25] showed that there exist instances for which the NS algorithm (with the pivot rule stated above) needs an exponential number of iterations. Orlin [20] 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 our paper. For a more elaborate discussion of the NS algorithm we refer to Ahuja et al. [1].

2 Upper Bound for the MMCC Algorithm

In this section we show an upper bound of $O(mn^2 \log(n) \log(\phi))$ for the expected number of iterations that the MMCC algorithm needs starting from the initial residual network $G_{\tilde{f}}$ for the feasible starting flow $\tilde{f}$ for flow network $G = (V, E)$. Note that we assumed in Section 1.1 that G is simple and that E does not contain a pair (u, v) and (v, u) of reverse edges. This implies that for each pair of nodes $u, v \in V$, there is always at most one edge from u to v and at most one edge from v to u in any residual network G_f. We first show that the number of cycles that appears in at least one residual network G_f for a feasible flow f on G, is bounded by $(n+1)!$, where $n = |V|$.

Lemma 2.1. *The total number of cycles that appears in any residual network G_f for a feasible flow f on G, is bounded by $(n+1)!$.*

We next show that the probability that any particular cycle has negative mean cost close to 0 can be bounded. In the rest of this section, $\varepsilon > 0$.

Lemma 2.2. *The probability that an arbitrary cycle C has mean cost $\mu(C) \in [-\varepsilon, 0[$ can be bounded by $n\varepsilon\phi$.*

Corollary 2.3. *The probability that there exists a cycle C with $\mu(C) \in [-\varepsilon, 0[$ is at most $(n+1)!n\varepsilon\phi$.*

Lemma 2.4. *If none of the residual networks G_f for feasible flows f on G contain a cycle C with $\mu(C) \in [-\varepsilon, 0[$, then the MMCC algorithm needs at most $mn\lceil \log_2(1/\varepsilon) \rceil$ iterations.*

Theorem 2.5. *The expected number of iterations that the MMCC algorithm needs is $O(mn^2 \log(n) \log(\phi))$.*

3 Lower Bound for the MMCC Algorithm

3.1 General Lower Bound

In this section 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, which occurs after a flow 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 how to choose the initial network, budgets, and starting flow such that G is the first residual network.

We now describe how to construct G given n, m, and ϕ. In the following, we assume $\phi \geq 64$. If ϕ is smaller than 64, the lower bound on the number of iterations reduces to $\Omega(m)$ and a trivial instance with $\Theta(n)$ nodes and $\Theta(m)$ edges will require $\Omega(m)$ iterations. 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 $k_x = k_w$ 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 $G = (\mathcal{V}, \mathcal{E})$ as follows.

- $\mathcal{V} = \{a, b, c, d\} \cup U \cup V \cup W \cup X$, 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 edge (u_i, v_j) has capacity 1 and cost interval $[0, 1/\phi]$.
- E_a contains the edges (a, u_i), E_b contains the edges (u_i, b), E_c contains the edges (c, v_i), and E_d contains the edges (v_i, d) $(i = 1, \ldots, n)$. All these edges have infinite capacity and cost interval $[0, 1/\phi]$.
- E_w contains the edges (d, w_i) and (w_i, a) $(i = 1, \ldots, k_w)$. An edge (d, w_i) has capacity m and cost interval $[0, 1/\phi]$. An edge (w_i, a) has capacity m and cost interval $[-2^{2-2i}, -2^{2-2i} + 1/\phi]$.
- E_x contains the edges (b, x_i) and (x_i, c) $(i = 1, \ldots, k_x)$. An edge (b, x_i) has capacity m and cost interval $[0, 1/\phi]$. An edge (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 edges of the types (w_i, a) and (x_i, c) have a cost interval that corresponds to negative edge costs. The residual network with these

negative edge costs can be obtained by having the following original instance (before computing a flow satisfying the budget requirements): All nodes, edges, costs and capacities are the same as in G, except that instead of the edges of type (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 edges of type (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 $k_x m$, the nodes of the types 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 that sends m units from a to each node of type w_i and from c to each node of type x_i then we obtain the initial residual network G.

We now show that the MMCC algorithm needs $\Omega(m \log(\phi))$ iterations for the initial residual network G. First we make some basic observations. The 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 cost 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 cost strictly less than $1/\phi$, cycles will never include more edges with cost at least $-1/\phi$ than necessary. In addition, since the edges of type (w_i, a) and (x_i, c) are saturated in the order cheapest to most expensive, none of these edges will ever be included in reverse direction in the minimum-mean cycle. The above observations lead to three candidate types for the minimum-mean cycle: cycles of type (d, w_i, a, u, v, d), of type (b, x_i, c, v, u, b), and of type $(d, w_i, a, u, b, x_j, c, v, d)$. 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 cycle types. 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 cost than a cycle C_2, we assume that all edges in C_1 take the highest cost in their cost interval, while all edges in C_2 take the lowest cost in their cost interval (an edge that appears in both C_1 and C_2 can even take its highest cost in C_1 and its lowest cost in C_2 in the analysis).

Lemma 3.1. *The cycle $C_1 = (d, w_i, a, u, v, d)$ has lower mean cost than the cycle $C_2 = (b, x_i, c, v, u, b)$.*

Lemma 3.2. *The cycle $C_1 = (b, x_i, c, v, u, b)$ has lower mean cost than the cycle $C_2 = (d, w_{i+1}, a, u, v, d)$.*

Lemma 3.3. *The cycle $C_1 = (d, w_i, a, u, v, d)$ has lower mean cost than the cycle $C_2 = (d, w_i, a, u, b, x_i, c, v, d)$.*

Lemma 3.4. *The cycle $C_1 = (b, x_i, c, v, u, b)$ has lower mean cost than the cycle $C_2 = (b, x_i, c, v, d, w_{i+1}, a, u, b)$.*

The above observations and lemmas allow us to determine the number of iterations that the MMCC algorithm needs for residual network G.

Theorem 3.5. *The MMCC algorithm needs $m(k_w + k_x)$ iterations for residual network G, independent of the realization of the edge costs.*

Proof (sketch). Using Lemma 3.1, Lemma 3.2, Lemma 3.3, and Lemma 3.4 we show that the first m iterations cancel cycles of type (d, w_1, a, u, v, d), the next m iterations cancel cycles of type (b, x_1, c, v, u, b), the next m iterations cancel cycles of type (d, w_2, a, u, v, d), etc. The total number of iterations is therefore $m(k_w + k_x)$.

The instance G and Theorem 3.5 allow us to state a lower bound on the number of iterations that the MMCC algorithm needs in the smoothed setting.

Theorem 3.6. *For every n, every $m \in \{n, n+1, \ldots, n^2\}$, and every $\phi \leq 2^n$, 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.*

3.2 Lower Bound for ϕ Dependent on n

In Section 3.1 we considered the setting where ϕ does not depend on n. 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 show that for every $n \geq 4$ and every $m \in \{n, \ldots, n^2\}$ there exists an instance with $\Theta(n)$ nodes, $\Theta(m)$ edges, and $\phi = \Theta(n^2)$ for which the MMCC algorithm needs $\Omega(mn)$ iterations.

The initial residual network H that we use to show our bound is very similar to the initial residual network G that was used to show the bound in Section 3.1. Below we describe the differences. We set $\phi = 400000n^2$. The constant 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 $\{w_1, \ldots, w_n\}$ and the node set X now consists of n nodes $\{x_1, \ldots, x_n\}$.
- 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 edges, all with infinite capacity and cost interval $[0, 1/\phi]$. Edges (a, u_i) are replaced by edges (a_2, u_i) with infinite capacity and cost interval $[0, 1/\phi]$. Edges (w_i, a) are replaced by edges (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 edges, all with infinite capacity and cost interval $[0, 1/\phi]$. Edges (c, v_i) are replaced by edges (c_2, v_i) with infinite capacity and cost interval $[0, 1/\phi]$. Edges (x_i, c) are replaced by edges (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 edges (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

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

As in Section 3.1, there are three candidate types for the minimum-mean cost cycle: cycles of type (d, w, a, u, v, d), cycles of type (b, x, c, v, u, b), and cycles of type $(d, w, a, u, b, x, c, v, d)$. Again we assume worst-case realizations of the edge costs and compare the mean costs of cycles of the different types in a series of lemmas.

Lemma 3.7. *The cycle $C_1 = (d, w_i, a, u, v, d)$ has lower mean cost than the cycle $C_2 = (b, x_i, c, v, u, b)$.*

Lemma 3.8. *The cycle $C_1 = (b, x_i, c, v, u, b)$ has lower mean cost than the cycle $C_2 = (d, w_{i+1}, a, u, v, d)$.*

Lemma 3.9. *The cycle $C_1 = (d, w_i, a, u, v, d)$ has lower mean cost than the cycle $C_2 = (d, w_i, a, u, b, x_i, c, v, d)$.*

Lemma 3.10. *The cycle $C_1 = (b, x_i, c, v, u, b)$ has lower mean cost than the cycle $C_2 = (b, x_i, c, v, d, w_{i+1}, a, u, b)$.*

The above lemmas allow us to determine the number of iterations that the MMCC algorithm needs for initial residual network H.

Theorem 3.11. *The MMCC algorithm needs $2mn$ iterations for initial residual network H, independent of the realization of the edge costs.*

Initial residual network H and Theorem 3.11 allow us to state a lower bound for the number of iterations that the MMCC Algorithm needs in the smoothed setting for large ϕ.

Theorem 3.12. *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.*

4 Lower bound for the Network Simplex Algorithm

In this section we provide a lower bound of $\Omega(m \cdot \min\{n, \phi\} \cdot \phi)$ for the number of iterations that the NS algorithm requires in the setting of smoothed analysis. The instance of the MCF problem that we use to show this lower bound is very similar to the instance used by Brunsch et al. [3] to show a lower bound on the number of iterations that the SSP algorithm needs in the smoothed setting. The differences are that they scaled their edge costs by a factor of ϕ, which we do not, that we add an extra path from node s to node t, and that the budgets of the nodes are defined slightly differently. We can show that every non-degenerate iteration of the NS algorithm for our instance corresponds with an iteration of the SSP algorithm for the instance of Brunsch et al. Because of space constraints we omit the analysis and only provide our main result.

Theorem 4.1. *For every n, every $m \in \{n, \dots, n^2\}$, and every $\phi \leq 2^n$ there exists a flow network with $\Theta(n)$ nodes and $\Theta(m)$ edges, and an initial spanning tree for which the Network Simplex algorithm needs $\Omega(m \cdot \min\{n, \phi\} \cdot \phi)$ non-degenerate iterations with probability* 1.

5 Discussion

In Section 4 we showed 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 that Brunsch et al. [3] showed for the SSP algorithm. 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, 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 conditions. 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 [15] 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.

There is a gap between our smoothed lower bound of $\Omega(m \log(\phi))$ (Section 3.1) for the number of iterations that the MMCC algorithm requires and our smoothed upper bound of $O(mn^2 \log(n) \log(\phi))$. 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 3.2) is another indication that this is likely possible.

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. Tobias Brunsch, Kamiel Cornelissen, Bodo Manthey, Heiko Röglin, and Clemens Rösner. Smoothed analysis of the successive shortest path algorithm. Computing Research Repository 1501.05493 [cs.DS], arXiv, 2015. Preliminary version at SODA 2013.

4. 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.
5. George B. Dantzig. *Linear programming and extensions*. Rand Corporation Research Study. Princeton Univ. Press, Princeton, NJ, 1963.
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. *J. ACM*, 36(4):873–886, October 1989.
10. Masao Iri. A new method for solving transportation-network problems. *Journal of the Operations Research Society of Japan*, 3(1,2):27–87, 1960.
11. William S. Jewell. Optimal flow through networks. *Operations Research*, 10(4):476–499, 1962.
12. Richard M. Karp. A characterization of the minimum cycle mean in a digraph. *Discrete Mathematics*, 23(3):309 – 311, 1978.
13. Morton Klein. A primal method for minimal cost flows with applications to the assignment and transportation problems. *Management Science*, 14(3):205–220, 1967.
14. Bernhard Korte and Jens Vygen. *Combinatorial Optimization: Theory and Algorithms*. Springer Publishing Company, Incorporated, 1st edition, 2007.
15. Péter Kovács. Minimum-cost flow algorithms: An experimental evaluation. *Optimization Methods and Software*, 30(1):94–127, 2015.
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. James B. Orlin. A polynomial time primal network simplex algorithm for minimum cost flows. *Math. Program.*, 77:109–129, 1997.
21. 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.
22. Daniel A. Spielman and Shang-Hua Teng. Smoothed analysis of algorithms: Why the simplex algorithm usually takes polynomial time. *J. ACM*, 51(3):385–463, 2004.
23. 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.
24. Éva Tardos. A strongly polynomial minimum cost circulation algorithm. *Combinatorica*, 5(3):247–256, 1985.
25. Norman Zadeh. A bad network problem for the simplex method and other minimum cost flow algorithms. *Mathematical Programming*, 5(1):255–266, 1973.