Smoothed Analysis of the Successive Shortest Path Algorithm*

Tobias Brunsch[1], Kamiel Cornelissen[2], Bodo Manthey[2], and Heiko Röglin[1]

[1]University of Bonn, Department of Computer Science, Germany.
[2]University of Twente, Department of Applied Mathematics, Enschede, The Netherlands.

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]. Plenty of pseudo-polynomial, polynomial, and strongly polynomial algorithms have been developed for the minimum-cost flow problem over the last fifty years. The fastest known strongly polynomial algorithm up to now is the Enhanced Capacity Scaling algorithm due to Orlin [10] and it 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 book of Ahuja, Magnanti, and Orlin [1].

Zadeh [14] showed that the Successive Shortest Path (SSP) algorithm has an exponential worst-case running time. Contrary to this, the worst-case running time of the strongly polynomial Minimum-Mean Cycle Canceling (MMCC) algorithm is $O(m^2n^2 \min\{\log(nC), m\})$ [11]. Here, C denotes the maximum edge cost. 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

*This research was supported by ERC Starting Grant 306465 (BeyondWorstCase) and NWO grant 613.001.023. It was presented at SODA 2013.

worst-case running times do not inevitably perform poorly in practice. An experimental study of Király and Kovács [7] 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. 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 [12] 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 because 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. We refer to [9] for a recent survey.

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 $G = (V,E)$, the edge capacities $u(e) \in \mathbb{R}^+$, and the balance values $b(v) \in \mathbb{R}$ of the nodes – indicating how much of a resource a node requires ($b(v) < 0$) or offers ($b(v) > 0$) – are chosen adversarially. 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. *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.

In practice, an instance of the minimum-cost flow problem is usually first transformed to an equivalent instance with only one *source* (a node with positive balance value) s and one *sink* (a node with negative balance value) t. The SSP algorithm then starts with the empty flow f_0. In each iteration i, it computes the shortest path P_i from the source s to the sink t in the residual network and maximally augments the flow along P_i to obtain a new flow f_i. The algorithm terminates when no $s-t$ path is present in the residual network.

Theorem 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 2 is due to Jewell [6], Iri [5], and Busacker and Gowen [4]. 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.

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, we denote the empty flow, i.e., the flow that assigns 0 to all edges e. 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.

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 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.

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

[6] William S. Jewell. Optimal flow through networks. *Oper. Res.*, 10(4):476–499, 1962.

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

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

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

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

[11] 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.

[12] 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.

[13] 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.

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