

# Path Trading: Fast Algorithms, Smoothed Analysis, and Hardness Results

André Berger<sup>1</sup>, Heiko Röglin<sup>2</sup> and Ruben van der Zwaan<sup>1</sup>

<sup>1</sup> Maastricht University, The Netherlands  
{a.berger,r.vanderzwaan}@maastrichtuniversity.nl  
<sup>2</sup> University of Bonn, Germany  
heiko@roeglin.org

**Abstract.** The Border Gateway Protocol (BGP) serves as the main routing protocol of the Internet and ensures network reachability among autonomous systems (ASes). When traffic is forwarded between the many ASes on the Internet according to that protocol, each AS selfishly routes the traffic inside its own network according to some internal protocol that supports the local objectives of the AS. We consider possibilities of achieving higher global performance in such systems while maintaining the objectives and costs of the individual ASes. In particular, we consider how *path trading*, i.e. deviations from routing the traffic using individually optimal protocols, can lead to a better global performance. Shavitt and Singer (“Limitations and Possibilities of Path Trading between Autonomous Systems”, INFOCOM 2010) were the first to consider the computational complexity of finding such path trading solutions. They show that the problem is weakly NP-hard and provide a dynamic program to find path trades between pairs of ASes.

In this paper we improve upon their results, both theoretically and practically. First, we show that finding path trades between sets of ASes is also strongly NP-hard. Moreover, we provide an algorithm that finds all Pareto-optimal path trades for a pair of two ASes. While in principal the number of Pareto-optimal path trades can be exponential, in our experiments this number was typically small. We use the framework of smoothed analysis to give theoretical evidence that this is a general phenomenon, and not only limited to the instances on which we performed experiments. The computational results show that our algorithm yields far superior running times and can solve considerably larger instances than the previous dynamic program.

## 1 Introduction

The Border Gateway Protocol (BGP) serves as the main routing protocol on the top level of the Internet and ensures network reachability among autonomous systems (ASes). When traffic is forwarded from a source to a destination, these ASes cooperate in order to provide the necessary infrastructure needed to ensure the desired services. However, ASes do

also compete and therefore follow their individual strategies and policies when it comes to routing the traffic within their own network. Such locally preferable routing decisions can be globally disadvantageous. Particularly, the way how one AS forwards traffic and through which ingress node another AS may therefore receive the traffic can make a huge difference in the costs for that other AS. Behaving selfishly usually means that an AS routes its traffic according to the least expensive route, also known as hot-potato routing, without regarding the costs of the next AS in the BGP path. This is supported by strong evidence by Teixeira et al. [17].

Quite a number of protocols have been suggested that require the exchange of information and coordination in order to overcome global suboptimality while at the same time improving the costs for each individual AS [6, 7, 18]. Recently, Shavitt and Singer [13] considered the case where ASes might be willing to *trade* traffic in such a way that the costs for both ASes do not increase w.r.t. the hot-potato routing, and term this problem *path trading*. They prove that for two ASes the problem of deciding whether there is a feasible path trade is weakly NP-hard. Further, they develop an algorithm based on dynamic programming to find the “best” trading between a pair. Lastly, they give experimental evidence that path trading can have benefits to autonomous systems.

In this paper we extend their work in the following way. We show that path trading is also strongly NP-hard when an arbitrary number of ASes is considered. This justifies the approach taken by Shavitt and Singer as well as the approach taken in this paper to concentrate on path trades between pairs of ASes. We propose a new algorithm for finding path trades between pairs of ASes that is based on the concept of *Pareto efficiency*. We have implemented both, our algorithm and the algorithm of Shavitt and Singer, and tested them on real Internet instances stemming from [12]. Besides the added advantage that our algorithm obtains *all* Pareto-optimal path trades, it is very fast and has low memory consumption. As the problem is NP-hard, we cannot expect that the algorithm performs well on all possible inputs. However, in order to support the experimental results we consider our algorithm in the framework of *smoothed analysis*, which was introduced in 2001 by Spielman and Teng [15] to explain why many heuristics with a bad worst-case performance work well on real-world data sets. We show that even though there are (artificial) worst-case instances on which the heuristic performs poorly, it has a polynomial expected running time on instances that are subject to small random perturbations. After its introduction, smoothed analysis has been applied in many different contexts (see [16] for a nice survey).

Finding path trades can be viewed as an optimization problem with multiple objectives that correspond to the costs of the different ASes. A *feasible path trade* is then a solution that is in every objective at least as good as the hot-potato routing. We say that such a path trade *dominates* the hot-potato routing if it is strictly better in at least one objective. This brings us to the well-known concept of *Pareto efficiency* or *Pareto optimality* in multiobjective optimization: A solution is called *Pareto-optimal* if it is not dominated by any other solution, that is, a solution is Pareto-optimal if there does not exist another solution that is at least as good in all criteria and strictly better in at least one criterion. We call the set of Pareto-optimal solutions *Pareto set* or *Pareto curve* for short.

Then the question of whether there is a feasible path trade can simply be formulated as the question whether the hot-potato routing is Pareto-optimal or not. This immediately suggests the following algorithm to find a feasible path trade: Enumerate the set of Pareto-optimal solutions, and then either output that there is no path trade if the hot-potato routing belongs to the Pareto set, or output a Pareto-optimal solution that dominates the hot-potato routing if it is not Pareto-optimal. Also, finding the Pareto set gives the flexibility to choose a solution based on preference. While some solutions might offer great global gain, these trade-offs might be unreasonable from a fairness perspective.

The aforementioned algorithm only works when the Pareto set is small because otherwise the computation becomes too time consuming. Our experiments show that the number of Pareto-optimal path trades is indeed small and that despite the NP-hardness result by Shavitt and Singer we can solve this problem efficiently in practice for two ASes.

For path trading between an arbitrary number of ASes, however, there is little hope for obtaining such a result: We show that our strong NP-hardness result implies that this problem cannot be solved efficiently even in the framework of smoothed analysis.

*Related Work.* The potential benefits of collaboration between neighboring ASes and the necessary engineering framework were first introduced by Winick et al. [18]. They consider the amount of information that needs to be shared between the ASes in order to perform mutually desirable path trades and how to limit the effect of path trades between neighboring ASes on the global flow of traffic. The first heuristics for path trading to improve the hot-potato routing were evaluated by Majahan et al. [7]. Majahan et al. also developed a routing protocol that provides evidence that path trading can improve global efficiency in Internet routing. Other

related work in the area of improving the global performance while maintaining the objectives of the different ASes has been done by Yang et al. [19], Liu and Reddy [6], and by Quoitin and Bonaventure [9]. Since ASes usually compete, one cannot expect them to reveal their complete network and cost structure when it comes to coordinating the traffic between the ASes. This aspect is considered in the work by Shrimali et al. [14], using aspects from cooperative game theory and the idea of Nash bargaining. Goldenberg et al. [4] develop routing algorithms in a similar context to optimize global cost and performance in a multihomed user setting, which extends previous work in that area [1, 3, 11].

## 2 Model and Notation

The model is as follows. We have the Internet graph  $G = (V, E)$ , where every vertex represents a point/IP-address. Further, there are  $k$  ASes and the vertex set  $V$  is partitioned into mutually disjoint sets  $V_1, \dots, V_k$ , where  $V_i$  represents all points in AS  $i$ . We denote by  $E_i$  all edges within AS  $i$ , that is, the set of edges  $E$  is partitioned into  $E_1, \dots, E_k$ , and the set of edges between different ASes. The graph  $G$  is undirected, and each edge  $e \in E$  has a length  $\ell(e) \in \mathbb{R}_{\geq 0}$ . The traffic is modeled by a set of *requests*  $R$ , where each request is a triple  $(s, t, c)$ , where  $s \in V$  and  $t \in V$  are source and sink nodes, respectively, and  $c \in \mathbb{R}_{\geq 0}$  is the cost of the corresponding request. The BGP protocol associates with each request a sequence of ASes which specifies the order in which the request has to be routed through the ASes. Since most of the paper is about the situation between *two* ASes, we leave this order implicit. The cost of routing a request with cost  $c$  through edge  $e$  is  $\ell(e) \cdot c$ . For simplicity, the costs of routing a packet between two ASes are assumed to be zero, but each request can be routed at most once from an AS to the next AS. The input for PATH TRADING consists of the graph  $G$  and requests as described previously. We denote by  $n$  the number of nodes in  $V$ .

For a given graph  $G$  and a request  $(s, t, c)$  we say that a path  $P$  is *valid* if it connects  $s$  to  $t$  and visits the ASes in the order that is associated with this request by the BGP protocol. This means, in particular, that every valid path goes through every AS at most once. A solution to PATH TRADING is a mapping that maps each request  $(s, t, c) \in R$  to a valid path from  $s$  to  $t$  in graph  $G$ . Let us assume that the requests in  $R$  are  $(s_1, t_1, c_1), \dots, (s_r, t_r, c_r)$  and that the paths  $P_1, \dots, P_r$  have been chosen for these requests. Then AS  $i$  incurs costs on all edges in  $E_i$ , i.e., it incurs

a total cost of

$$\sum_{j=1}^r \left( c_j \cdot \sum_{e \in P_j \cap E_i} \ell(e) \right). \quad (1)$$

The *hot-potato route* of a request  $(s, t, c)$  is defined to be the concatenation of shortest path routes for the ASes it goes through. To be precise, assume that the BGP protocol associates the route  $i_1, \dots, i_m$  with  $s \in V_{i_1}$  and  $t \in V_{i_m}$  with this request. Then AS  $i_1$  sends the request from  $s$  to the vertex  $s_2 \in V_{i_2}$  that is closest to  $s$  along the shortest path. Then AS  $i_2$  sends the request from  $s_2$  to the vertex  $s_3 \in V_{i_3}$  that is closest to  $s_2$  along the shortest path, and so on. The complete hot-potato route for request  $(s, t, c)$  is then the concatenation of these paths. Note that the hot-potato route is not necessarily unique, and in the following we will assume that some hot-potato route is chosen for each request.

Consequently, the costs of the hot-potato routing that an AS  $i$  incurs are equal to Equation 1, where the paths  $P_1, \dots, P_r$  are the hot-potato paths. We call a solution to PATH TRADING a *path trade* and if the costs for all involved ASes are less or equal to their hot-potato costs, then we call it a *feasible path trade*. In the following, let  $[n]$  be the set of integers  $\{1, \dots, n\}$ . For a vector  $x \in \mathbb{R}^n$ , let  $x_i$  be the  $i$ -th component of  $x$ .

Due to space limitations the proofs of the complexity results (Theorems 1, 2, 4 and Corollary 2) are deferred to full version of this paper.

### 3 Complexity Results and Smoothed Analysis

Our first result is about the complexity of PATH TRADING and extends the weak NP-hardness result of Shavitt and Singer [13]. The proof uses a reduction from 3-PARTITION.

**Theorem 1.** *Finding a feasible path trade apart from the hot-potato routing is strongly NP-hard.*

Given the above theorem, in order to develop fast algorithms, we concentrate on path trading between two ASes, and will now present our algorithm for this case. As mentioned before, this algorithm is based on the concept of Pareto efficiency and it enumerates all Pareto-optimal path trades. In the worst case the number of Pareto-optimal solutions can be exponential, but our experiments suggest that on real-world data usually only a few solutions are Pareto-optimal. To give a theoretical explanation for this, we apply the framework of smoothed analysis. The algorithm is a dynamic program that adds the requests one after another, keeping track

of the Pareto-optimal path trades of those requests that have already been added.

For a request  $(s, t, c)$ , a path  $P$  from  $s$  to  $t$ , and  $i \in \{1, 2\}$ , we denote by  $C_i(P)$  the costs incurred by AS  $i$  due to routing the request along path  $P$ . To keep the notation simple, assume in the following discussion w.l.o.g. that  $s \in V_1$  and  $t \in V_2$ . We denote by  $\mathcal{P}(s, t)$  the set of all Pareto-optimal valid paths from  $s$  to  $t$ . Remember that a path is valid if it starts at  $s$ , terminates at  $t$ , and does not go back to  $V_1$  after leaving  $V_1$  for the first time. Such a path  $P$  belongs to  $\mathcal{P}(s, t)$  if there does not exist another valid path that induces strictly lower costs for one AS and not larger costs for the other AS than  $P$ . We assume that in the case that there are multiple paths that induce for both ASes exactly the same costs, only one of them is contained in  $\mathcal{P}(s, t)$ .

Let  $P \in \mathcal{P}(s, t)$  be some Pareto-optimal path and let  $v \in V_1$  be the *boundary node* at which the path leaves AS 1. Then the subpaths from  $s$  to  $v$  and from  $v$  to  $t$  must be shortest paths in AS 1 and AS 2, respectively. Otherwise,  $P$  cannot be Pareto-optimal. Hence, the number of Pareto-optimal paths in  $\mathcal{P}(s, t)$  is bounded from above by the number of boundary nodes of AS 1 that connect to AS 2. For each pair  $s \in V_1$  and  $t \in V_2$ , the set  $\mathcal{P}(s, t)$  can be computed in polynomial time.

Our algorithm first computes the set  $\mathcal{P}_1$  of Pareto-optimal path trades for only the first request  $(s_1, t_1, c_1)$ . This is simply the set  $\mathcal{P}(s_1, t_1)$ . Based on this, it computes the set  $\mathcal{P}_2$  of Pareto-optimal path trades for only the first two requests, and so on. Thus the elements in  $\mathcal{P}_i$  are tuples  $(P_1, \dots, P_i)$  where each  $P_j$  is a valid path for the  $j$ th request.

---

**Algorithm 1** Algorithm to compute the Pareto set

---

```

 $\mathcal{P}_1 = \mathcal{P}(s_1, t_1);$ 
for  $i = 2$  to  $r$  do
   $\mathcal{P}_i = \{(P_1, \dots, P_i) \mid (P_1, \dots, P_{i-1}) \in \mathcal{P}_{i-1}, P_i \in \mathcal{P}(s_i, t_i)\};$ 
  Remove all solutions from  $\mathcal{P}_i$  that are dominated by other solutions from  $\mathcal{P}_i$ .
  If  $\mathcal{P}_i$  contains multiple solutions that induce for both ASes exactly the same costs,
  then remove all but one of them.
end for
Return  $\mathcal{P}_r$ 

```

---

**Theorem 2.** For  $i \in [r]$ , the set  $\mathcal{P}_i$  computed by Algorithm 1 is the set of Pareto-optimal path trades for the first  $i$  requests. In particular, the set  $\mathcal{P}_r$  is the set of Pareto-optimal path trades for all requests. Algorithm 1 can be implemented to run in time  $O(n \log n \cdot \sum_{i=1}^r |\mathcal{P}_i| + nr|E| \log n)$

We start by reviewing a result due to Beier et al. [2] who analyzed the number of Pareto-optimal solutions in binary optimization problems with two objective functions. They consider problems whose instances have the following form: the set of feasible solutions  $S$  is a subset of  $\{0, \dots, F\}^n$  for some integers  $F$  and  $n$ , and there are two objective functions  $w^{(1)} : S \rightarrow \mathbb{R}$  and  $w^{(2)} : S \rightarrow \mathbb{R}$  that associate with each solution  $x \in S$  two weights  $w^{(1)}(x)$  and  $w^{(2)}(x)$  that are both to be minimized. While  $w^{(2)}$  can be an arbitrary function, it is assumed that  $w^{(1)}$  is linear of the form  $w^{(1)}(x) = w_1x_1 + \dots + w_nx_n$ .

In a worst-case analysis, the adversary would be allowed to choose the set of feasible solutions  $S$ , and the two objective functions  $w^{(1)}$  and  $w^{(2)}$ . Then it can easily be seen that there are choices such that the number of Pareto-optimal solutions is exponential. To make the adversary less powerful and to rule out pathological instances, we assume that the adversary cannot choose the coefficients  $w_1, \dots, w_n$  exactly. Instead he can only specify a probability distribution for each of them according to which it is chosen independently of the other coefficients. Without any restriction, this would include deterministic instances as a special case, but we allow only probability distributions that can be described by a density function that is upper bounded by some parameter  $\phi \geq 1$ .

We denote by  $f_i : \mathbb{R}_{\geq 0} \rightarrow [0, \phi]$  the probability density according to which  $w_i$  is chosen, and we assume that the expected value of  $w_i$  is in  $[0, 1]$ .

**Theorem 3 (Beier et al., [2]).** *For any choice of feasible solutions  $S \subseteq \{0, \dots, F\}^n$ , any choice of  $w^{(2)}$  and any choice of density functions  $f_1, \dots, f_n$ , the expected number of Pareto-optimal solutions is bounded by  $O(\phi n^2 F^2 \log F)$ .*

Now we formulate our problem in terms of Theorem 3. For this, we assume that all requests have positive integer costs. Let  $F$  denote an upper bound on the maximal costs possible on any edge, e.g.,  $F = \sum_{(s,t,c) \in R} c$ . Let  $m = |E|$  and assume that the edges in  $E$  are ordered arbitrarily. Then each path trade leads to a cost vector  $x \in \{0, \dots, F\}^m$  where  $x_1$  denotes the total cost of all requests that use the first edge in  $E$ ,  $x_2$  denotes the total cost of all requests that use the second edge in  $E$ , and so on. If two solutions lead to the same cost vector, then it suffices to remember one of them, and hence, we can assume that the set of possible path trades can essentially be described by the set  $S \subseteq \{0, \dots, F\}^m$  of possible cost vectors. Given such a cost vector  $x \in \{0, \dots, F\}^m$ , we can express the cost  $w^{(1)}(x)$  of the first AS simply as  $\sum_{e \in E_1} \ell(e)x_e$ . The costs of the

second AS can be defined analogously, and so it looks that Theorem 3 directly applies when we perturb all edge lengths  $\ell(e)$  of edges  $e \in E_1$  as these edge lengths are the coefficients in the linear objective function  $w^{(1)}$ . However, there is a small twist. In Theorem 3 all coefficients in the linear objective function  $w^{(1)}$  are chosen randomly. Our objective function, however, does not contain terms for the edges  $e \in E_2$ . Or with other words the coefficients are 0 for these edges. If we apply Theorem 3 directly, then also these zero coefficients would be perturbed, which would destroy the combinatorial structure of the problem as then suddenly the cost of the first AS would depend on what happens on edges  $e \in E_2$ .

To avoid this side effect, we remodel the feasible region  $S$ . As argued before, each solution leads to a cost vector  $x \in \{0, \dots, F\}^m$ , but now we care only about the part of the vector for  $E_1$ . Let us define  $m' = |E_1| \leq m$ . Then each solution leads to a cost vector  $x \in \{0, \dots, F\}^{m'}$  that contains only the costs of the first AS. Now of course different solutions can lead to the same vector  $x$  if they differ only in the way how the traffic is routed in the second AS. However, Theorem 3 allows completely general objective functions  $w^{(2)}$ , which we exploit by defining  $w^{(2)}(x)$  for a vector  $x \in \{0, \dots, F\}^{m'}$  to be the smallest cost for the second AS that can be achieved with any solution whose cost vector for the first AS results in  $x$ . This formulation implies the following corollary.

**Corollary 1** *Given a path trading instance in which the edge lengths  $\ell(e)$  for all  $e \in E_1$  are randomly chosen according to probability distributions that satisfy the same restrictions as those in Theorem 3, the expected number of Pareto-optimal solutions is bounded by  $O(\phi m^2 F^2 \log F)$ .*

Given that the expected number of Pareto-optimal solutions is small, we still have to show that Algorithm 1 computes the Pareto curve in expected polynomial time. This will be established by the following Corollary.

**Corollary 2** *Algorithm 1 computes the Pareto curve in expected time  $O(\phi n m^2 \log n \cdot r F^2 \log F)$ .*

The reason that we concentrate our efforts on path trading between two ASes was the hardness result in Theorem 1. We can extend this result and also show that there is no hope for PATH TRADING with an arbitrary number of ASes to be solvable efficiently in the framework of smoothed analysis.

**Theorem 4.** *There is no smoothed polynomial time algorithm for PATH TRADING with an arbitrary number of ASes, unless  $NP \subseteq BPP$ .*

## 4 Evaluation

In this section we present the experimental results about the performance of our algorithm on the IP-level Internet graph from DIMES [12]. We compare it, in particular, with the performance of the dynamic program used by Shavitt and Singer, and we answer the following questions:

- *How fast can we compute the Pareto curve?* Algorithm 1 is very fast and scales well.
- *How robust are both algorithms?* When we add random costs to all requests, the running time of Algorithm 1 does not increase much. This suggests even in environments with large costs, Algorithm 1 will perform well. The running time of the dynamic program directly depends on the costs, and it becomes quickly infeasible to compute solutions for even small instances.
- *How many ASes are involved in path trading?* In our experiments we see that for low amount of requests, roughly 60% of all ASes engage in path trading.

The answers to these questions are, of course, depending on the assumptions we made. As Shavitt and Singer, we assume that traffic is symmetric, i.e., the number of requests sent from AS  $A$  to AS  $B$  is the same as the number of requests sent from AS  $B$  to AS  $A$  for every pair of ASes. This assumption is not necessarily true, but it is common and used, e.g., in [13], [7], and [14]. One could imagine that in real networks requests are not evenly spread, but are more concentrated between popular ASes for example. Still, even for low amounts of requests there was substantial gain for the ASes involved. This indicates that even if the traffic between two ASes is asymmetric, they have a good chance of gaining from path trading as long as there is non-zero traffic in both directions. Our second assumption is that each request between two ASes has to be routed between two nodes that are chosen uniformly at random from these ASes. Our third assumption is that all edges have length 1, i.e., the number of hops is used to measure the costs of an AS for routing a request. By absence of real data, we feel that this a reasonable and common assumption. We first perform experiments in which every request has costs 1 and then repeat the experiments with requests with randomly chosen costs. These experiments demonstrate that our method is robust against changes of the costs of the requests. In the following subsection we present the details of the experimental setup. The algorithm by Shavitt and Singer is named Algorithm 2. Then we show and discuss the experimental results.

## 4.1 Experimental Setup

We assume that traffic is symmetric. We used the Internet graph from DIMES [12], and we assumed that every edge length is one, and all packets have cost one. So, the costs of a request are the number of hops on the route. The whole Internet graph from DIMES contains roughly 27 thousand ASes and 3.5 million nodes. Of all ASes, 1276 ASes and 4348 AS pairs were sufficiently connected: These pairs had edges between them and more than one boundary node. To determine participation, we simulated a low number of requests for each sufficiently connected pair, to find out whether a small or a large fraction of ASes are involved in path trading.

For both algorithms, we need to calculate shortest paths beforehand. Because of the large number of possible routings, many shortest paths need to be computed. This was all done as part of the preprocessing, and all shortest paths were stored in a hash-table for fast access for both algorithms. In the following, this time is not included in the running times of the algorithms.

To measure how many ASes could benefit from path trading, we simulated 5 requests for each of the 4348 sufficiently connected AS pairs in either direction. For comparing performance and robustness we selected a subset of 15 AS pairs arbitrarily among the AS pairs that benefited from path trading in the first experiment where 5 requests were sent in either direction.

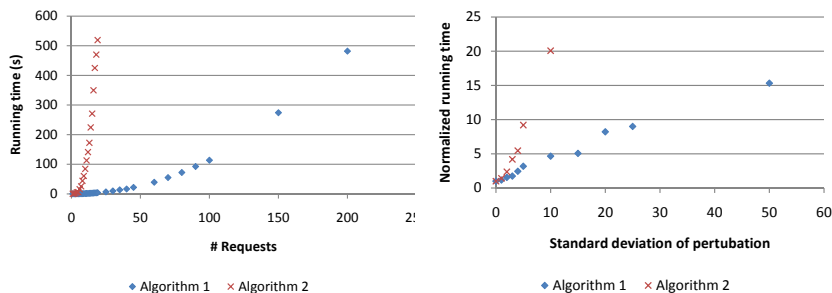
To get some idea about how robust both algorithms are, we increased the costs of the requests. For each request  $(s, t, c)$  we set  $c$  as  $c = 1 + X$ , where  $X$  is a random variable, normally distributed with mean 0 and standard deviation  $\sigma$ . Further,  $X$  was capped at 0 from below and at 10 from above for  $\sigma \in \{1, 2, 3, 4, 5\}$ . For  $\sigma = 10$ ,  $X$  was capped at 0 from below and at 20 from above. This was done to prevent extremely long running times for the dynamic program. For  $\sigma \in \{15, 20, 25, 50\}$ ,  $X$  was only capped at 0 from below. All numbers were rounded to the nearest integer. We simulated 10 requests in either direction, for each of the pairs.

## 4.2 Experimental Results

**Performance** Table 1 shows a comparison of the running times of Algorithm 1 and Algorithm 2. The running times are the total of the running time over the 15 selected pairs in seconds. In these ASes, roughly 37% of the costs were saved by path trading. As can be seen, the running time of Algorithm 2 quickly becomes very high.

# Requests	Algorithm 1 (s)	Algorithm 2 (s)	Ratio
1	0.02	0.09	1: 4.5
5	0.19	6.04	1: 31.79
10	1.09	84.31	1: 77.35
15	2.38	270.87	1:113.81
19	4.01	519.27	1:129.49

**Table 1.** The performance of Algorithm 1 compared to Algorithm 2.



**Fig. 1.** (a) Running times of both algorithms compared. (b) The normalized running times of both algorithms plotted against magnitude of the perturbations.

The memory usage is dominated by the number of Pareto optimal solutions, and each Pareto optimal solution is represented as a tuple of two integers. Figure 1(a) shows a graphical comparison of both algorithms. Not only is Algorithm 1 fast for small amounts of requests, it can handle up to ten times more requests in the same time as Algorithm 2.

**Robustness** We find that the running time of both Algorithm 1 and Algorithm 2 is influenced by larger request costs, but not to the same degree. Figure 1(a) shows the running times of both algorithms. As can be seen, the running time of Algorithm 2 quickly spirals out of control. Algorithm 1 stays computable, although the running time does increase. Figure 1(b) displays the running times normalized with regard to the running time without perturbations for both Algorithm 1 and 2.

The step increase of the running time of Algorithm 2 comes at no surprise as the dynamic program is directly dependent on the costs of the routing, and not on the number of different choices or the complexity

of the network. The experiments show that our algorithm is significantly more robust against non-uniform request costs.

*Acknowledgements.* The authors would like to thank Tobias Brunsch for proofreading this manuscript and for his helpful comments.

## References

1. A. Akella, B. Maggs, S. Seshan, A. Shaikh, and R. Sitaraman. A measurement-based analysis of multihoming. SIGCOMM, pages 353–364, 2003.
2. R. Beier, H. Röglin, and B. Vöcking. The smoothed number of Pareto optimal solutions in bicriteria integer optimization. IPCO, pages 53–67, 2007.
3. R. Dai, D. O. Stahl, and A. B. Whinston. The economics of smart routing and QoS. NGC, pages 318–331, 2003.
4. D. K. Goldenberg, L. Qiu, H. Xie, Y. R. Yang, and Y. Zhang. Optimizing cost and performance for multihoming. SIGCOMM, pages 79–82, 2004.
5. Donald Knuth. The Art of Computer Programming, Volume 3: Sorting and Searching, Third Edition. Addison-Wesley, 1997.
6. Y. Liu and A. L. N. Reddy. Multihoming route control among a group of multihomed stub networks. Computer Comm., 30(17):3335–3345, 2007.
7. R. Mahajan, D. Wetherall, and T. Anderson. Negotiation-based routing between neighboring ISPs. NSDI, pages 29–42, 2005.
8. G. L. Nemhauser and Z. Ullmann. Discrete dynamic programming and capital allocation. Management Science, 15(9):494–505, 1969.
9. B. Quoitin and O. Bonaventure. A cooperative approach to interdomain traffic engineering. EuroNGI, 2005.
10. H. Röglin and S.-H. Teng. Smoothed Analysis of Multiobjective Optimization. FOCS, pages 681–690, 2009.
11. P. Sevcik and J. Bartlett. Improving user experience with route control. Technical Report NetForecast Report 5062, NetForecast, Inc., 2002.
12. Y. Shavitt and E. Shir. DIMES: let the Internet measure itself. ACM SIGCOMM Computer Communication Review, 35(5):71–74, 2005.
13. Y. Shavitt and Y. Singer. Limitations and Possibilities of Path Trading between Autonomous Systems. INFOCOM, 2010.
14. G. Shrimali, A. Akella, and A. Mutapcic. Cooperative interdomain traffic engineering using nash bargaining and decomposition. INFOCOM, pp. 330–338, 2007.
15. D. A. Spielman and S.-H. Teng. Smoothed analysis of algorithms: why the simplex algorithm usually takes polynomial time. Journal of the ACM, 51(3):385–463, 2004.
16. D. A. Spielman and S.-H. Teng. Smoothed analysis: an attempt to explain the behavior of algorithms in practice. Communic. of the ACM, 52(10):76–84, 2009.
17. R. Teixeira, A. Shaikh, T. Griffin, and J. Rexford. Dynamics of hot-potato routing in IP networks. SIGMETRICS, pages 307–319, 2004.
18. J. Winick, S. Jamin, and J. Rexford. Traffic engineering between neighboring domains. Technical report, 2002.
19. Y.R. Yang, H. Xie, H. Wang, A. Silberschatz, A. Krishnamurthy, Y. Liu, and E.L. Li. On route selection for interdomain traffic engineering. IEEE Network, 19(6):20–27, 2005.