Optimizing OSPF/IS-IS Weights in a Changing World

Bernard Fortz

Institut d'Administration et de Gestion
Université Catholique de Louvain
Louvain-la-Neuve
Belgium
fortz@qant.ucl.ac.be

Mikkel Thorup

AT&T Labs-Research
Shannon Laboratory
Florham Park, NJ 07932
USA
mthorup@research.att.com

Intended for IEEE JSAC Special Issue on Advances in Fundamentals of Network Management see
http://www.argreenhouse.com/society/J-SAC/Calls/fundamentals.html

Abstract

A system of techniques is presented for optimizing OSPF/IS-IS weights for intra-domain routing in a changing world, the goal being to avoid overloaded links. We address predicted periodic changes in traffic as well as problems arising from link failures and emerging hot-spots.

Keywords: SPF, OSPF, IS-IS, traffic engineering, traffic management, local search, combinatorial optimization.

1 Introduction

The optimization goals of traffic engineering are to enhance the performance of IP traffic while utilizing network resources economically. In this paper, our focus is on *optimizing OSPF/IS-IS traffic routing so as to make more efficient use of network resources in scenarios of changes to traffic and networks.*

Shortest Path First (SPF) protocols such as Open Shortest Path First (OSPF) [22] or Intermediate System-Intermediate System (IS-IS) [9] are the most commonly used intra-domain internet routing protocols today. Traffic is routed along shortest paths to the destination. The weights of the links, and thereby the shortest path routes, can be changed by the network operator. A simple default weight setting suggested by Cisco [11] is to make the weight of a link inversely proportional to its capacity. As an alternative to OSPF/IS-IS, the more flexible Multi-Protocol Label Switching (MPLS) protocol has been suggested [5, 24]. MPLS is not yet widely deployed, but in principle, it would allow arbitrary routing in networks.

Our general objective in this paper is to route demands through an OSPF/IS-IS based network so as to avoid congestion in terms of link loads exceeding capacities with resulting packet loss and back-off in TCP.

In the context of a *fixed* network with a fixed known demand matrix this problem has already been addressed experimentally in [18] with real and synthetic data, showing that we can find weight settings supporting 50%-110% more demands than Cisco's defaults inverse-capacity-weights, and get within a few percent of the best possible with general routing, including MPLS. Similar positive findings have been reported in [23], [19], and [7]. Here, the demand matrix could be based on concrete measurements, as described in [15] (see also [6, 12]), but could also be based on concrete service level agreements (SLAs).

However, as stipulated in [3], *demand matrices and networks change*. The obvious idea for dealing with change is to just reset the weights using the above mentioned techniques. However, as we shall see shortly, there are several reasons why one should *avoid weight changes* as much as possible. Our target in this paper is to match the above mentioned improvements for fixed networks and demand matrices in scenarios of changing networks and demand matrices, changing as few weights as possible.

Why weight changes are bad There are several reasons why weight changes are to be avoided as much as possible.

(i) Weights are often not set centrally, and if hundreds of weights are to be changed, there is a good chance of human errors.

(ii) It takes time for a network to flood information about a new weights, for new shortest path routes to be computed, and for the routing tables to get updated. Changing a lot of weights, this could create temporary chaos in a network with packets arriving out of order, degrading the performance of TCP. Also, the changes may impact the routes advertised to other autonomous systems whose routing may then also experience fluctuations.

(iii) A human network operator is responsible for the routing and has to approve the changes. The network operator may have several requirements to the routing that are not specified for

the weight optimizing algorithm, e.g. that certain demands have to be routed along certain links. It is very hard for a human to check the consequences of hundreds of weight changes, but the consequences of a few weight changes should be easy to grasp.

Above, (i) may be resolved soon [20], but the other points remain valid.

Good performance with few weight changes Motivated by the problems in changing weights, our first contribution is a technique for optimizing weights, changing as few weights as possible.

The technique was applied to a real weight setting of the AT&T IP backbone, and it was found that increasing a single weight from 1024 to 1025 reduced the max-utilization by 8%. Here, the *utilization* of a link is the ratio of its load over its capacity, so utilization above 100% means an overloaded link. The max-utilization is the maximum utilization over all links in the network. Thus, from the perspective of max-utilization, the weight change was worth an 8% increase in link capacities. Checking the impact of a single weight change is relatively easy, and executing it is much cheaper than buying and installing new links with higher capacity.

Our technique was also tried on the experimental networks from [18], including a proposed AT&T IP backbone as well as the synthetic 2-level graphs of Zegura et al. [28, 29, 10]. Here it was found that if we started with Cisco's suggested default link weights inversely proportional to link capacities, using at most 10 weight changes, we could support around 50% more demands, achieving about 2/3 of the gains reported in [18] with a complete change of all weights.

Of particular interest, we considered the problem of reestablishing performance after a local change such as a link-failure or a new hot-spot. That is, our starting point was a set of weights optimized for a network with a given demand matrix. We then simulated all possible single link-failures and hot-spots, and discovered that even in the worst cases, three weight changes sufficed to get within a few percent of the best possible with a complete dynamic change of all weights.

We note that improved routing may cause TCP to send packets more aggressively, thus changing the demand matrix. Allowing TCP to send more traffic thanks to less loaded links and packet loss is in itself considered positive, and here we ignore the more complicated interaction. We note, however, that we in response to demand changes from TCP could apply the weight changes iteratively.

Periodic daily changes Even on a daily basis there are large structural differences between day and evening traffic. However, this pattern is relatively similar for different days, that is, the traffic undergoes quite predicted periodic changes on a daily basis [15, 16].

The most obvious idea for dealing with daily changes would be to use the above techniques to adapt the weights to the changes. However, even a few daily weight changes are best avoided.

We propose here to seek *one* weight setting that is good for *all* the typical periodic changes happening during a day. To deal with periodic changes, we suggest optimizing the weight setting based on a few representative demand matrices, such as one day and one evening matrix. Our basic idea is that if one set of weights works well for some representative demand matrices, it works as well for all demand matrices dominated ($\leq$ in each entry) by convex combinations of these representative demand matrices, thus allowing us to cover an infinite space of demand matrices rather than just a few singular points. This combined with general robustness to noise gives us a chance of addressing all the periodic changes happening throughout a day.

For our test instances, we succeeded in finding single weight settings for pairs of structurally different demand matrices that simultaneously for each individual demand matrix gave routing close to the optimum for general routing (including MPLS) for that demand matrix. That is, our fixed weight setting is competitive with ideal dynamic routing with unlimited power to adapt independently to each demand matrix.

This finding is somewhat surprising in that it is easy to construct pairs of demand matrices so that however we fix the routing, even with MPLS, it is going to be bad for at least one of the demand matrices.

Our successful optimization over multiple demand matrices is also interesting for *differentiated services* with Quality of Service (QoS) constraints. Here we want to find one weight setting that on the one hand satisfies certain performance guarantees for high priority customers, and on the other hand gives good best effort service for the general traffic. Having one demand matrix for the high priority customers and another for the regular traffic, we can apply the above mentioned techniques.

Summary of contribution This is the first paper addressing the problem of finding good weight settings for OSPF/IS-IS without the freedom to change all weights. We believe that respecting the problem in changing weights takes us from a nice classical style combinatorial optimization problem to something of true practical relevance.

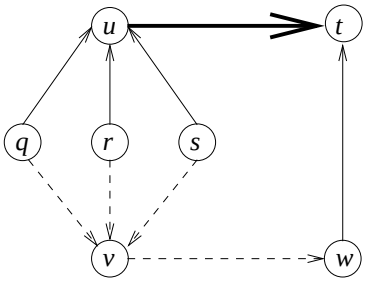
Our technique for tuning a weight setting with as few weight changes as possible is a novel general adaptation of the classical local search approach of combinatorial optimization [1], and we expect it to find many other applications in areas where parameter changes are undesirable. From a programming perspective, the adaptation had the advantage that we could largely reuse our previous local search code for a good setting of all weights [18].

Concerning periodic changes, our main new idea is *not* to do the obvious thing of precomputing weight changes for a transition between, say, day and evening, but to strive for one weight setting good for the whole day.

The value of our techniques is demonstrated experimentally on synthetic and real networks, showing that we can improve significantly over standard default weight settings, and in fact get close to the best dynamic routing with the more general MPLS scheme. As pointed out in [4, 17, 18], it is trivial to construct networks and demand matrices for which the best OSPF/IS-IS routing is worse than the best MPLS routing. Also, in this paper, we present a negative example with two demand matrices that do not have a good common routing, even with MPLS. Thus, our positive results cannot be achieved in general, but they indicate that the negative examples are unlikely to dominate in real networks.

In combination, our techniques provide an efficient tool for OSPF/IS-IS traffic management in a changing world. It is now an integrated part of AT&T's NetScope/Bravo traffic engineering tool for IP networks [14]. Note that our work may also be applicable to MPLS traffic engineering when label-switched paths or tunnels become logical links with weights like the physical links in OSPF/IS-IS.

Contents First, in §2, we define our exact model and objectives. Next, in §3, we present our technique for optimizing with few weight changes. In §4, this technique is applied in connection



Link	Default unit weights		Increasing weight of overloaded link		Optimal global single change	
	Weight	Load	Weight	Load	Weight	Load
(q,u)	1	1	1	0.5	1	1
(r,u)	1	1	1	0.5	1	1
(s,u)	1	1	1	0.5	3	0
(u,t)	1	**3**	2	1.5	1	**2**
(q,v)	1	0	1	0.5	1	0
(r,v)	1	0	1	0.5	1	0
(s,v)	1	0	1	0.5	1	1
(v,w)	1	0	1	1.5	1	1
(w,t)	1	1	1	**2.5**	1	**2**

Figure 1: The need for global weights optimization

with link failures and hot-spots. Then, in §5, we describe and evaluate our approach with multiple demand matrices. Finally, we have some concluding remarks in §6.

2 Models and experimental test-bed

This section presents and discusses the framework from [18] used to evaluate weight settings for a given network and demand matrix. Whereas [18] kept the network and demand matrix fixed, we are going to show how to deal with changes in the subsequent sections.

2.1 The general routing problem

Optimizing the use of existing network resources can be seen as a general routing problem defined as follows. We are given a capacitated directed graph $G = (N, A, c), A \subseteq N \times N, c := (c_a)_{a \in A}$, whose nodes and arcs represent routers and the capacitated links between them, and a demand matrix D that, for each pair (s,t) of nodes, tells us how much traffic flow we need to send from s to t. We refer to s and t as the source and the destination of the demand. Many of the entries of D may be zero, and in particular, $D[s,t]$ should be zero if there is no path from s to t in G. A routing solution specifies for each source-destination pair how the demanded traffic should flow in the network. The load ℓ_a on an arc a is then the total traffic flow through the arc, including the contributions from each source-destination pair.

In the left side of Figure 1 is a concrete example of a network where each of the nodes q, r, s, and w demand a flow of 1 to t. In the displayed routing solution, the flows just follow shortest paths to the destination, and thereby a load of 3 is accumulated on link (u,t) whereas all other links have loads 0 or 1. The table on the right will be discussed in the next subsection.

For real instances of the problem, additional complicating constraints such as nodes forbidden for transit traffic or point-to-multi-point demands arise [15]. These kind of constraints can be integrated by modifying the graph including artificial links, but these constraints do not affect the methods and results presented here and are left out for the sake of clarity.

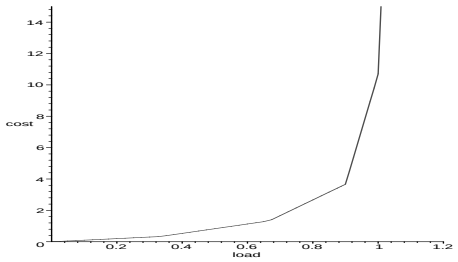


Figure 2: Arc cost $\phi(\ell_a, 1)$ as a function of load ℓ_a with capacity $c_a = 1$.

So far, we have been rather vague about our objective of "avoiding overloaded arcs", and we will now define some more exact objectives. Recall that the utilization of an arc a is the load divided by the capacity, i.e. ℓ_a/c_a, and a link is overloaded if the utilization exceeds 100%. The max-utilization is the maximum utilization over all links. Thus, if all links in Figure 1 had capacity 2, the max-utilization would be $3/2 = 150\%$.

Minimizing the max-utilization as in [19] is a very natural and intuitive objective for routing. In our example, we can reduce the max-utilization to 1 by sending the traffic from s via v and w to t instead of via u. We will consider the max-utilization in this paper, but it suffers from allowing a single bottle-neck, e.g. an ingress link from another domain over which we have no control, to dominate the whole picture. Also, it doesn't penalize using very long detours. To get a measurement considering the whole network, we consider cost functions of the form

$$\Phi = \sum_{a \in A} \phi(\ell_a, c_a)$$

summing a cost $\phi(\ell_a, c_a)$ from each arc a depending on the relation between the load ℓ_a and the capacity c_a. In [23] they use a queuing theory style link cost function $\phi(\ell_a, c_a) = \ell_a/(c_a - \ell_a)$. With this function, it is more expensive to send flow along arcs whose loads approach capacity, which is what we want. However, the function doesn't deal with overloaded links, and in reality, a single overloaded link does not take down a whole network. To overcome this problem, we resorted to a piece-wise linear approximation of $\ell_a/(c_a - \ell_a)$, defined with heavy penalty for overloaded links. More precisely, for some fixed capacity c_a, we define $\phi(x, c_a)$ as the continuous function with $\phi(0, c_a) = 0$ and derivative in the load x of

$$\phi'(x, c_a) = \begin{cases} 1 & \text{for} \quad 0 \le x/c_a < 1/3, \\ 3 & \text{for} \quad 1/3 \le x/c_a < 2/3, \\ 10 & \text{for} \quad 2/3 \le x/c_a < 9/10, \\ 70 & \text{for} \quad 9/10 \le x/c_a < 1, \\ 500 & \text{for} \quad 1 \le x/c_a < 11/10, \\ 5000 & \text{for} \quad 11/10 \le x/c_a < \infty. \end{cases} \tag{1}$$

The arc cost function ϕ is illustrated in Figure 2. Generally it is cheap to send flow over an arc with a small utilization ℓ_a/c_a. The cost increases progressively as the utilization approaches 100%, and explodes when we go above 110%.

Because of the explosive increase in cost as loads exceed capacities, our objective typically implies that we keep the max-utilization below 1, or at least below 1.1, if at all possible.

The exact coefficients are not important. We tried many variations in the objective function and found that this did not change the essence of our results. More importantly, the routing solutions found were very robust to changes in the objective function. In particular, when optimizing routings for Φ, our solutions tended to also do very well with respect to max-utilization, and based on this we conjecture that they are good with respect to any "reasonable" cost function.

The piece-wise linearity of our cost function has the advantage that using a Linear Programming (LP) solver, we can find the optimal solution to the general routing problem with no limitations to how we can distribute the flow between the paths. We can then compare ourselves against this unrealistic ideal to see how competitive we are with *any* other approach, including MPLS.

A problem in the current formulation of Φ is that it does not provide a universal measure of congestion. With the max-utilization, it is clear for any routing in any network that we have a problem if it exceeds 100%, and we would like a similar universal cut-off for our summed link-costs. To achieve this, we use a normalized cost function[1]

$$\Phi^* = \Phi / \Psi$$

where Ψ is the cost we would have had if all flow was sent along hop-count shortest paths and all loads matched the capacities. More formally, if $\Delta(s,t)$ is the hop-count distance between s and t, $\Psi = \sum_{(s,t)\in V^2}(D[s,t]\cdot\Delta(s,t)\cdot\phi(1,1))$. Note that for a given network and demand matrix, the division by Ψ doesn't affect which routings are considered good. Summing up, $\Phi^* \geq 1$ implies that we are performing as badly as if all flows where along hop-count shortest paths with loads matching the capacities. The same cost can, of course, also stem from some loads going above capacity and others going below, or by flows following longer detours via less utilized arcs. Nevertheless, it is natural to say that a routing *congests* a network if $\Phi^* > 1$.

2.2 OSPF/IS-IS routing

As mentioned earlier, this paper focuses on routing with OSPF [22] and IS-IS [9], which are the most commonly used intra-domain internet routing protocols today. The network operator assigns a weight to each link, and shortest paths from each router to each destination are computed using these weights as lengths of the links. In each router, the next link on all shortest paths to all possible destinations is stored in a table, and a flow arriving at the router is sent to its destination by splitting the flow between the links that are on the shortest paths to the destination. The details of the splitting/tie-breaking depends on the configuration of the router. In this paper we assume a hash-based splitting, as used in the AT&T IP network: if a router has multiple outgoing links on shortest paths to a destination, packets are assigned an outgoing link based on a hash function of some information in their header. The hashing ascertains that packets from the same flow follow the same path, which is important for the packets to arrive in the order they were transmitted. The hash based splitting generally results in a roughly even split, and for simplicity, we assume that the split is exactly even.

In [17, 18], it was found that Cisco's [11] suggested default of setting weights inversely proportional to the capacity was as good or better than setting all weights to 1, or using physical distance.

[1]The normalization from [18] was defined differently so that it was $\phi(1,1) = 10\frac{2}{3}$ times bigger.

This justifies that inverse-capacity is the only default weight setting that we compare ourselves against in the current paper.

2.3 The need for global optimization

A naive approach to the problem of trying to improve a given routing by modifying only one or few weights would be to just increase the weight of overloaded links, thus avoiding the need for sophisticated tools as presented in this paper. (Un)fortunately, such a strategy is often insufficient to provide the best results. To illustrate this, reconsider the network of Figure 1 assuming that all capacities are 2. Suppose that we want to send one unit of flow from q, r, s and w to t. The table in Figure 1 presents the loads resulting from three different weight settings. For units weights, we get a maximal load of 3 on link (u,t). If we wish to decrease this maximal load by increasing the weight of this heavily loaded link, the best solution is to set the weight to 2. We then get that the flows coming from q, r and s are split along the two possible paths, and the maximal load becomes 2.5 on link (w,t). If we instead look for the best change over all links, we easily see that setting the weight of link (s,u) to 3 changes the routing for the flow from s to t, but not those from q and r to t, and the maximal load in the network is thereby decreased to 2, matching the capacity.

Note that in this optimal solution, the shortest paths are unique, hence that we made no use of the even splitting feature. Rather it just singles out part of the traffic from s and redirects it. A similar pattern was found in many of the optimized weight settings from [18]. A somewhat similar finding is reported in [25], where the authors, in a slightly different context, show that the best routing strategy usually is obtained with a limited number of different paths.

2.4 Test instances

Our basic experimental networks and demand matrices are the same as in [17, 18]. We have a proposed AT&T IP backbone with 90 nodes, 274 arcs and projected demands. Also, we have a synthetic 2-level networks produced using the generator GT-ITM [28], based on a model of Calvert, Bhattacharjee, Daor, and Zegura [29, 10]. This model places nodes in a unit square, thus getting a distance $\delta(x,y)$ between each pair of nodes. These distances lead to random distribution of 2-level graphs, with arcs divided in two classes: *local access* arcs and *long distance* arcs. Arc capacities were set equal to 200 for local access arcs and to 1000 for long distance arcs. Many networks were generated and tested, also including Waxman graphs [27] and random graphs, but for space reasons, we only present experiments over one 2-level graph with 50 nodes and 148 arcs. Results for other networks are consistent with those reproduced here and follow the same patterns.

The above synthetic network model does not include a model for the demands. Inspired by classical entropy models for urban traffic [26], we decided to model the demands as follows. For each node x, we pick two random numbers $o_x, d_y \in [0,1]$. Further, for each pair (x,y) of nodes we pick a random number $c_{(x,y)} \in [0,1]$. Now, if the Euclidean distance (L_2) between x and y is $\delta(x,y)$, the demand between x and y is

$$\alpha o_x d_y c_{(x,y)} e^{-\delta(x,y)/2\Delta} \tag{2}$$

Here α is a parameter and Δ is the largest Euclidean distance between any pair of nodes. Above, the o_x and d_x model that different nodes can be more or less active senders and receivers, thus modeling hot spots on the net. Because we are multiplying three random variables, we have a

quite large variation in the demands. The factor $e^{-\delta(x,y)/2\Delta}$ implies that we have relatively more demand between close pairs of nodes, yet the distance on its own never has an impact bigger than a factor $\sqrt{e} = 1.648\ldots$. In our experiments, we also tried not using the factor $e^{-\delta(x,y)/2\Delta}$, and the results were essentially unchanged.

In our experiments, we generated one demand matrix for each network, then we scaled it (by multiplying each entry by a constant) at different levels to obtain different total demands. All our results are reported for these increasing total demands, which allows to measure, for a given routing, at which level of total demand congestion occurs for the given topology and demand pattern.

Our technique for few weight changes was also tested on the real AT&T IP backbone with its real weight setting and measured demands [15].

Discussion of test instances We will now briefly discuss our choice of synthetic data. On the positive side, for the network model itself, it has already been established that it gives good approximations to real networks in many aspects [29, 10]. Further, the similarity in results found in [18] between the proposed AT&T IP backbone and our synthetic networks indicate that the model is reasonably faithful for our purpose.

On the negative side, it has recently been found that the degree distribution for IP networks has heavy tails [13] (we note that [13] actually considers the Internet as a whole whereas we are only considering a single domain), and similar findings have been found for the demands [15], that is, we should have some nodes with very high degrees, and some very large demands. Our synthetic models do not provide such heavy tails.

Generating networks and demand matrices with heavy tails is in itself easy [2]. For the networks, one could even use some of the official networks available at [8]. However, in real networks, the heavy tails of the networks and demand matrices are highly dependent. Generally, heavy tails means structure. High degree nodes are placed at strategic positions relative to the structure of the expected demand matrix. Indeed, we have found that AT&T's network performs very poorly on a random demand matrix. A random heavy tail demand matrix placing a large demand in, say, Alaska, would be even worse. Thus, networks and demand matrices with heavy tails have to be tailored to each other to be realistic. So far, there is no established model for such tailored combination, and tailoring the input data ourselves would make our results less convincing.

The advantage of Zegura's networks is that they are constructed in a generic way not favoring any particular demands, and for that reason they work reasonably well for generic demand matrices such as ours.

Thus, we argue there is no obvious better choice of a model, given that, to claim good optimization results, we cannot convincingly construct a model by hand. However, for the sake of optimization work like ours, we see it as an important challenge to develop a single heavy tail model integrating both IP networks and demand matrices where the IP network is reasonably tuned for the demand matrix. In fact, as argued above, we think that such an integrated approach may give a better understanding of the heavy tails found in IP network, viewing the hight degree nodes as strategically positioned hops.

3 Few weight changes

In this section, we consider the problem of optimizing, making as few weight changes as possible. We apply our technique for few changes in two scenarios. One is when the weight setting has not previously been optimized, e.g. if the network is so far just run with inverse capacity weights following Cisco's recommendation [11]. The other one, described in Section 4, is in response to problematic changes in demands or topology like critical link failures or emerging hot-spots.

3.1 Technique for few changes

We want to make as few weight changes as possible. The local search from [17, 18] that we used for multiple demand matrices works with a single solution that is iteratively improved by small changes in the current solution. It typically performs a lot of iterations (5000 in our practical experiments), and therefore produces a solution completely different from the starting one. This can be seen as a depth-first search in the solution space, but since we want to make as few changes as possible, our approach should rather be a breadth first search.

Our heuristic for improving an input weight setting w_0 with as few weight changes as possible works as follows: first we consider about 1000 single weight changes to w_0, corresponding to about 5 weight changes for each arc in our largest networks. Instead of selecting only the best weight change as in [17, 18], we keep the 100 best weight changes in a family F of "best weight settings". The process is iterated with F instead of w_0: we consider 1000 single weight changes for each weight setting in F and a new F is selected containing the 100 best of the old weight settings in F and the about 100,000 new weight settings considered. After i iterations, including the start from w_0, the family consists of weight settings with up to i weight changes from w_0. The size of F corresponds to the breadth of our search. All the above numbers are just parameters that experimentally were found to give a good compromise between quality of solution and time. For the largest networks considered, 10 iterations took about 1 1/2 hours on a 194 MHz CPU SGI Challenge XL. If running time is an issue, one can, for example, reduce the family size or the number of iteration, or even better, buy a better faster computer.

This technique for few changes has, to our knowledge, not been used before. Its main interest is that it provides a general framework for optimizing with few changes that can be easily adapted for other applications. It has the advantage that if a local search such as our previous heuristic from [18] is available, the main ingredients such as the changes applied to one solution to get a neighbor of it, or the procedures to evaluate a new solution, can be reused in this new framework, saving a lot of implementation work.

3.2 Experiments with few changes

As our base experiment with few changes, we took the inverse-capacity weight setting (InvCap-OSPF) as a starting point. A concrete output for running the code for 3 iterations is presented in Figure 3.

It tells the user how much improvement can be obtained with 1, 2, and 3 changes and which changes to make. Further, it shows more precisely how the loads on the most utilized links change as more changes are applied.

```
Number of iterations set to 3.
Will profile top 4 utilized links.
Demand scaling factor set to 2.916967.
Data file:    experiments/hier100.dat

No input weights. Using InvCap as default.
Scaling factor       :    16113.563
InvCap               :        8.873 (1.210)
   Utilization profile:
        69:2.420e+02/2.000e+02=   1.210
       170:1.639e+02/2.000e+02=   0.820
       176:1.584e+02/2.000e+02=   0.792
        89:1.564e+02/2.000e+02=   0.782
Iteration     1      :        1.323 (0.820)
        w[69] : 5 -> 11
   Utilization profile:
       170:1.639e+02/2.000e+02=   0.820
        69:1.623e+02/2.000e+02=   0.811
        89:1.564e+02/2.000e+02=   0.782
        81:1.533e+02/2.000e+02=   0.766
Iteration     2      :        1.308 (0.811)
        w[69] : 5 -> 11
        w[170] : 5 -> 10
   Utilization profile:
        69:1.623e+02/2.000e+02=   0.811
        89:1.564e+02/2.000e+02=   0.782
        81:1.533e+02/2.000e+02=   0.766
       231:1.499e+02/2.000e+02=   0.750
Iteration     3      :        1.294 (0.782)
        w[100] : 5 -> 1
        w[69] : 5 -> 12
        w[170] : 5 -> 9
   Utilization profile:
        89:1.564e+02/2.000e+02=   0.782
        81:1.533e+02/2.000e+02=   0.766
       180:1.466e+02/2.000e+02=   0.733
        63:1.456e+02/2.000e+02=   0.728
Adaptive CPU time    :          221
Best found           :        1.294 (0.782)
```

Figure 3: Optimizing with few changes on 2-level graph with 100 nodes and 280 arcs.

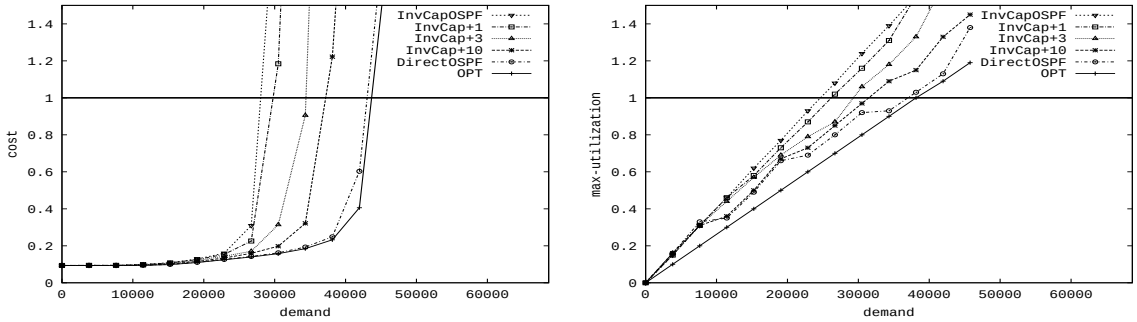


Figure 4: Few changes on AT&T's proposed backbone.

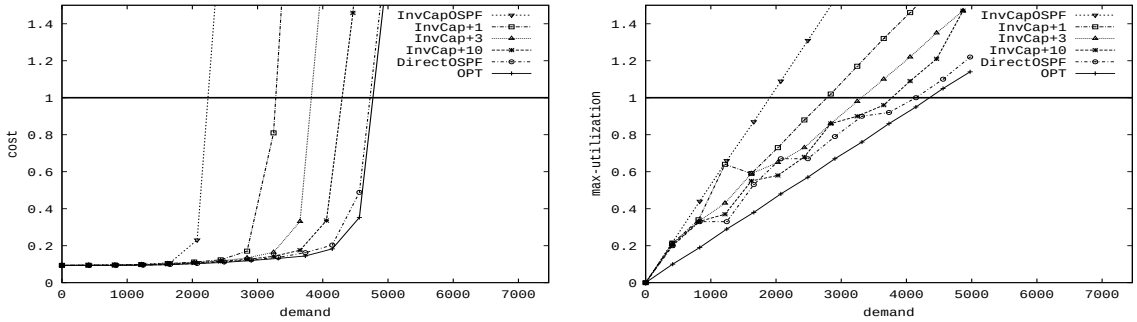


Figure 5: Few changes on 2-level graph.

The results of applying our technique are presented in Figures 4–5, with InvCap+k denoting that we applied k changes. DirectOSPF represents a weight setting, involving all weights, optimized with our local search from [18], and OPT is the solution of the general routing problem, including the possibilities with MPLS.

First, as in [17, 18], we note that all curves start off pretty flat, and then, quite suddenly, start increasing rapidly. This behavior follows our cost function that explodes when the load of a link reaches its capacity (cf. (1) and Figure 2).

The most interesting comparison between the different schemes is the amount of demand they can cope with before the network suddenly gets congested in the sense that its normalized cost exceeds 1. Here 1 change gives 10%–50%, 3 changes give 25%–75% while 10 changes give 35%–90%. Thus quite substantial improvements can be obtained with few changes.

The code was also run with on the real AT&T IP backbone with its real weight setting and measured demands. It was found that changing the weight of a single link from 1024 to 1025 improved the max-utilization by 8%. Further details on this are, however, proprietary.

4 Link failures and hot-spots

In this section, we consider the problem of link failures and emerging hot spots. We show that they are not normally problematic. However, there are a few critical cases, but for these, we reestablish good performance with at most 3 weight changes.

4.1 Link failures

For AT&T's proposed IP backbone, we tried all possible link-failures, and computed the routing both with InvCapOSPF, and with a weight setting optimized from before the link-failure (DirectOSPF). The results are depicted in Figure 6, giving the values, first without link-failures, then as average over all link-failures, and finally with worst-case link failures. Here, by worst-case, we mean the link whose deletion decreases the objective function the most for the given weight setting. This link is thus found independently for each weight setting considered.

For average link failures, we see that DirectOSPF performs 25% better than InvCapOSPF, which is still pretty good. However, for worst-case link failures for the proposed AT&T IP backbone in Figure 6, we see that our optimized weights do 65% worse than InvCapOSPF. Shortly we will turn this defeat into victory.

If we remove the worst-case link that caused the biggest problem for weight settings found by our DirectOSPF for the proposed AT&T IP backbone and re-optimize with as few weight changes as possible, as described in the last section, we obtain results depicted in Figure 7. Here OldOSPF denotes our optimized weight setting from before the link failure, and Old+k denotes the result of optimizing this weight setting with k changes. InvCapOSPF, DirectOSPF, and OPT are all based directly on the network without the bad link.

First observe that OldOSPF comes down and perform as well or better than InvCapOSPF for demands scaled around 11000 and 19000. The reason for this is that for the original weight settings generated for these two demand scalings, it was actually a different link that was bad. However, for all the other demand levels, it is the worst possible link we have deleted. This essentially means

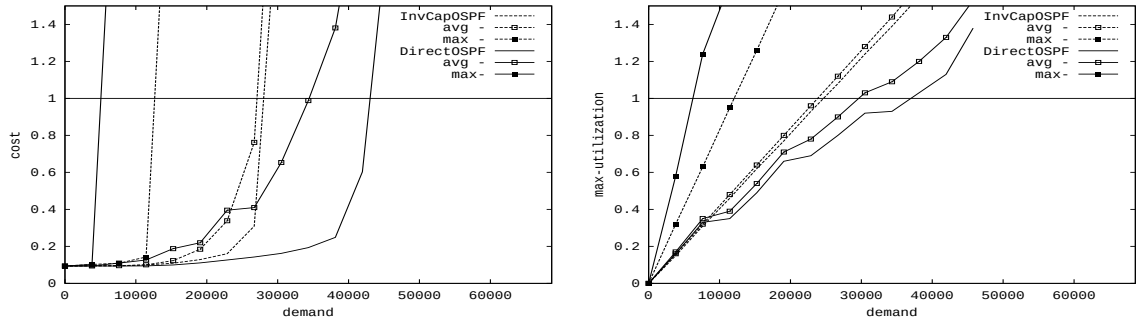


Figure 6: Link-failure on AT&T's proposed backbone.

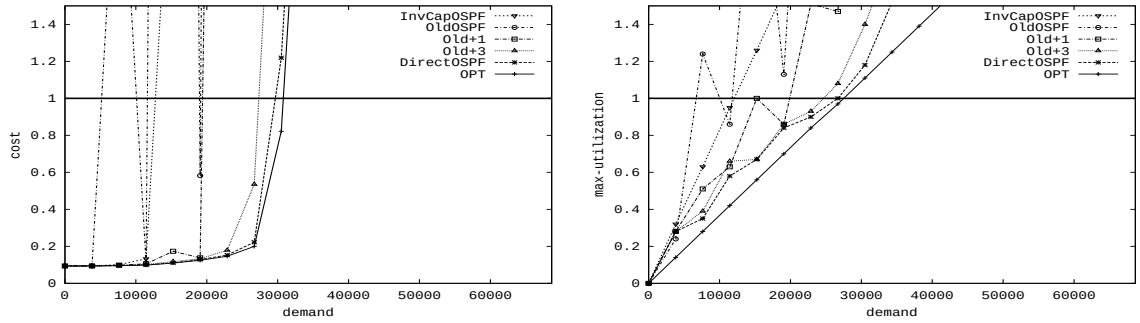


Figure 7: Resurrecting performance after worst possible link failure in AT&T's proposed backbone.

that our weight setting had some different choices in what links to make critical, and the choice happened to come out differently for two of the scalings. The jumps nearly disappear after just one weight change, which makes us beat InvCap everywhere.

What we now see is that after just 1 weight change, we do 60% better than InvCap and with 3 changes we do 160% better, getting within 10% of OPT.

4.2 New hot-spots

Our next experiments concern the developments of new hot-spots. The essential experiment is to take one router and multiply all incoming demands by a factor of 3. This corresponds to multiplying one d_x by 3 in (2), which again amounts to multiplying all entries in a column of the demand matrix by 3. This is likely to give more structural difference than just noise, turning links towards x into bottlenecks.

The results of the experiments are depicted in Figure 8. As for link failures, the average and the worst performance are obtained over hot-spotting all nodes, one at the time. We do this with the weight settings both from InvCapOSPF and DirectOSPF. We see that DirectOSPF performs 20% better than InvCapOSPF for average hot-spots, and 30% better for worst-case hot-spots. Without hot-spots, DirectOSPF performed 50% better than InvCapOSPF. Thus, we do lose some of our advantage over InvCapOSPF, but our optimized weight setting still provides a clear advantage

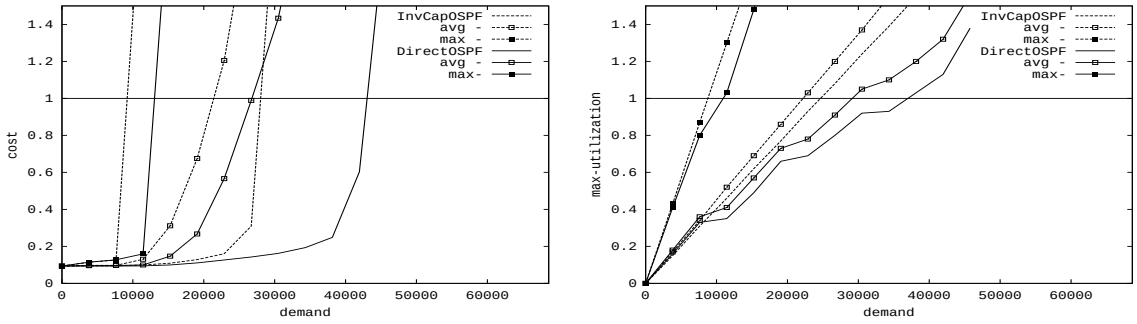


Figure 8: Hot-spot on AT&T's proposed backbone.

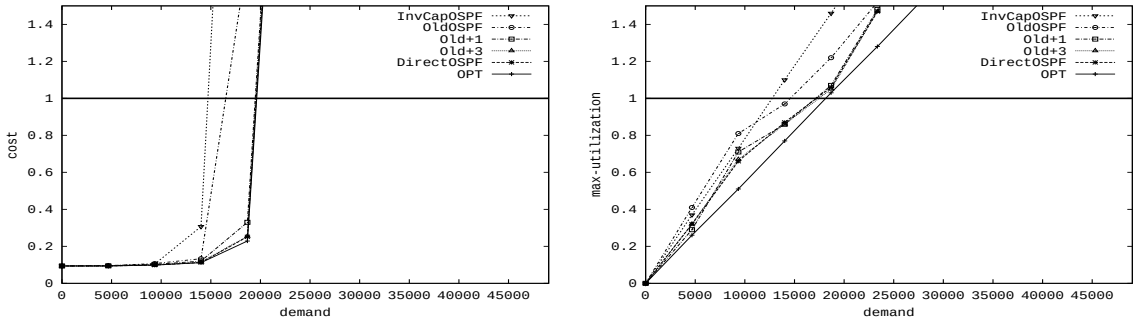


Figure 9: Resurrecting performance the worst hot spot in AT&T's proposed backbone.

over InvCapOSPF.

As for link-failures, we tried to take the worst possible hot-spot (in the sense that it deteriorates the most the objective function), and reestablish good performance. The result is depicted in Figure 9. The nice thing is that we only need a single weight change to get back very close to optimum, which is even better than for the worst possible link failure.

5 Multiple demand matrices

Our motivation for working with multiple demand matrices is the general experience from AT&T that traffic follows quite regular periods with a peak in the day and in the evening. The network operators do not want to change weights on a regular basis so we want just one weight setting which is good for the whole period. We then collect a peak demand matrix for the day and one for the evening. A weight setting performing good on both performs good on all convex combinations, and hence it has a good chance of performing well for the whole period.

5.1 An impossible example

In Figure 10, it is illustrated that if we have two fix one set of routes for two different demand matrices, this may force us to increase the max-utilization by 50% for one of them. In the example,

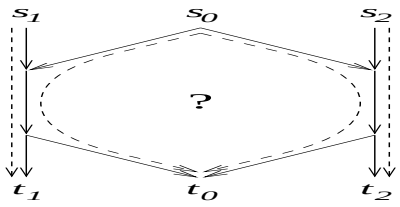


Figure 10: Bad choice

one demand matrix wants one unit of demand from s_0 to t_0 and from s_1 to t_1 while the other wants one unit of demand from s_0 to t_0 and from s_2 to t_2. However, any flow from s_0 to t_0 has to share a link with either flow from s_1 to t_1, or from s_2 to t_2. In each situation independently, we can get max-utilization 1, but if we have to fix the routing from s_0 to t_0, the best we can do is to split the s_0-t_0 flow evenly, getting a max-utilization of $3/2$ in both cases. As for the negative examples in [18], this worst-case example is far from real networks, and we will see that in practice, real networks perform much better.

5.2 Optimizing for multiple demand matrices

Given a network $G = (N, A, c)$ with several demand matrices $D_1, ..., D_k$, we want to find a single weight setting $w := (w_a)_{a \in A}$ which works well for all of them. In general, we will use $\ell_a(G, D, w)$ to denote the load on link a with network G, demand matrix D, and weight setting w. Similarly for our cost function, we have $\Phi(G, D, w) = \sum_a \Phi_a(\ell_a(G, D, w))$ with Φ_a as defined in (1).

Now consider a demand matrix D dominated by a convex combination of $D_1, ..., D_k$, that is $D \leq \alpha_1 D_1 + \cdots + \alpha_k D_k$ where $\alpha_1 + \cdots + \alpha_k = 1$. Here everything is understood to be entry-wise, so for all x, y, $D[x, y] \leq \alpha_1 D_1[x, y] + \cdots + \alpha_k D_k[x, y]$.

Since the routing for each source-destination pair is fixed by the weight setting w, for each arc $a \in A$, $\ell_a(G, D, w) \leq \alpha_1 \ell_a(G, D_1, w) + \cdots + \alpha_k \ell_a(G, D_k, w)$. In particular, it follows that the max-utilization for D is no worse that the worst max-utilization for the D_i. Further, since each arc cost function Φ_a is convex, $\Phi_a(\ell_a(G, D, w)) \leq \alpha_1 \Phi_a(\ell_a(G, D_1, w)) + \cdots + \alpha_k \Phi_a(\ell_a(G, D_k, w))$, and hence $\Phi(G, D, w) \leq \alpha_1 \Phi(G, D_1, w) + \cdots + \alpha_k \Phi(G, D_k, w)$. Thus, our weight setting w does no worse for D than for the worst of the D_i, neither with respect to our cost function Φ, nor with respect to max-utilization. Note that the same observation holds true with MPLS, as long as the routing for each source-destination pair is fixed.

From [17], we know that it is NP-hard even to approximate a good weight setting for a single fixed demand matrix, that is, unless NP=P, there cannot be any general method providing guaranteed good results. However, an efficient local search heuristic for the problem is suggested in [17, 18] that for a given network G and demand matrix D looks for a weight setting w that minimizes $\Phi(G, D, w)$.

To optimize simultaneously for several demand matrices $D_1, ..., D_k$, we simply modify the local

search heuristic to minimize

$$\Phi(G, D_1, \ldots, D_k, w) = \sum_{i \leq k} \Phi(G, D_i, w) \tag{3}$$

As in our original motivation for defining Φ, this has the the effect of penalizing highly loaded links, this time, for all the demand matrices instead of just one. Our negative theoretical example shows that we cannot in general hope for good results, but we can still hope to do well in practice.

5.3 Experiments for multiple demand matrices

In [17, 18] there was only one demand matrix D_1 for each network. Here we generated a second independent matrix D_2, using the same distribution, and scaled to have the same total demand as D_1. We call this demand matrix a twin of D_1. The difference between D_1 and D_2 models structural differences in traffic, with a complete change in who are the big senders and receivers, thus modeling, e.g., the difference between day and evening traffic. We also derived a noisy version $\tilde{D}_1$ of D_1 that will be discussed later.

We compared OPT, InvCapOSPF and DirectOSPF (obtained with our heuristic from [18]) against OSPF/IS-IS routings of D_1 using weight settings optimized relative to D_2 (TwinOSPF) and for D_1 and D_2 simultaneously (PairOSPF) as in (3). A symmetric set of experiments were performed with the roles of D_1 and D_2 interchanged. A good performance of PairOSPF on both D_1 and D_2 shows that we have successfully found one weight setting that compromised neither demand matrix.

The results of our experiments are presented in Figures 11–14 with different scalings of the demand matrices.

In the experiments, we see that DirectOSPF allows us to cope with 50%-110% more demand than the oblivious heuristics. Also, DirectOSPF is less than 2% from being able to cope with the same demands as the optimal general routing OPT.

If we look at TwinOSPF, we see that it is often at least as bad, or worse than InvCapOSPF. Nevertheless, PairOSPF does very well, never getting more than 10% worse than DirectOSPF and OPT. In particular, PairOSPF is always doing simultaneously well for both demand matrices it optimizes over.

Put in terms of our day-evening example, this means that if we just optimize relative to a peak demand matrix from the day, we cannot expect to gain anything for the evening. However, if we do simultaneous optimization for a day and evening demand matrix, we can hope do well on both. This combined with our robustness to noise (see below) and our automatic benefit for all convex combinations gives us a good chance of dealing well for typical daily periodic changes with a single weight setting.

One may wonder why things work so well in spite of simple negative examples like the one presented in Figure 10. A simple explanation comes from the fact that even if a bad example occurs as a subgraph of the network, real networks are usually large enough that we can steer the traffic around such a small bottle-neck even with limited control over the routing. Indeed, in our experiments it usually turned out that only a small number of links where overloaded, so our bottlenecks were local, and making just a few good weight changes allowed us to smooth out these local discrepancies.

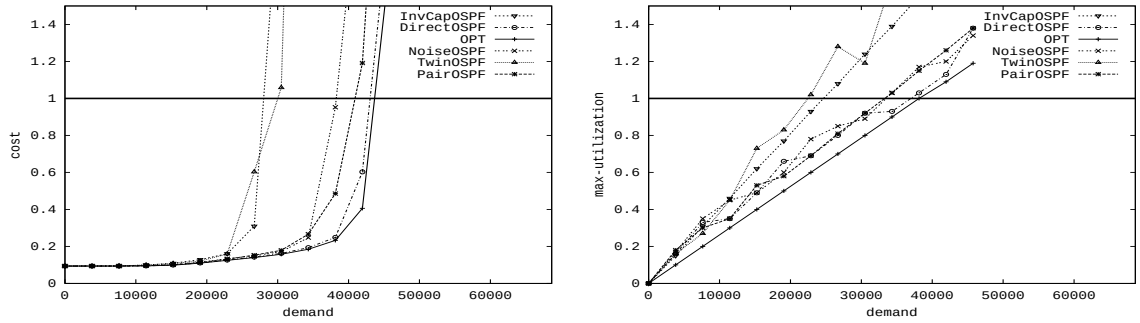


Figure 11: Simultaneous optimization on AT&T's proposed backbone and scaled projected demands (D_1).

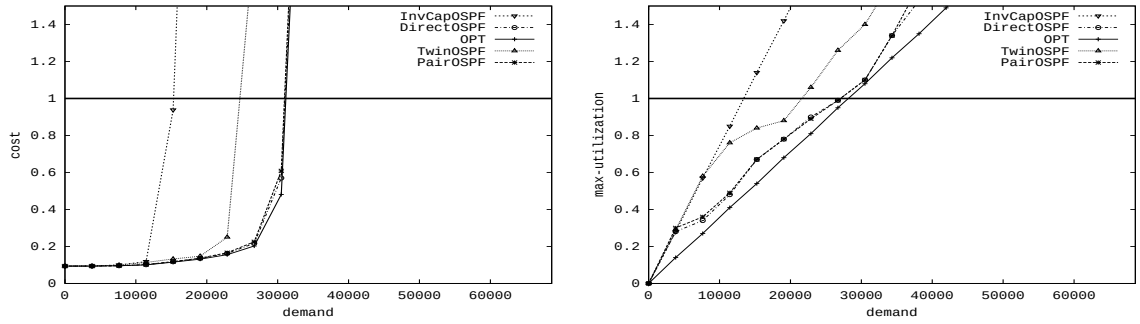


Figure 12: Simultaneous optimization on AT&T's proposed backbone and synthetic twin demands (D_2).

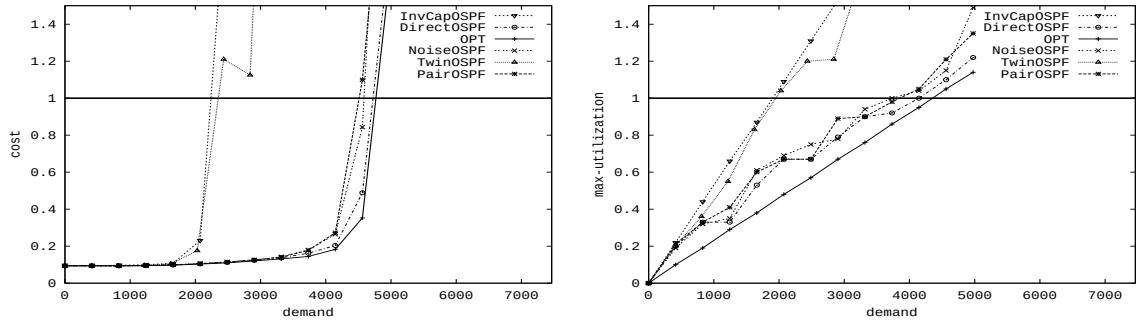


Figure 13: Simultaneous optimization on 2-level graph (D_1).

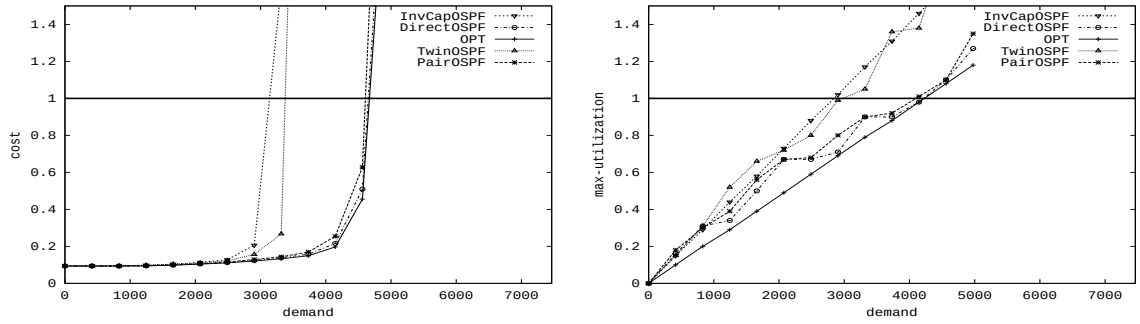


Figure 14: Simultaneous optimization on 2-level graph and twin demands (D_2).

5.4 Robustness to noise

We derived a noisy version $\tilde{D}_1$ of D_1 by multiplying each entry by a random number between 0 and 2. On the average this changes each entry by 50% without changing the expected total demand. The discrepancy between D_1 and $\tilde{D}_1$ could represent problems of getting exact measures [15], or general fluctuations in traffic.

The results of our experiments with noise are also presented in Figures 11 and 13, where we compared DirectOSPF against routings of D_1 using weight settings optimized relative to $\tilde{D}_1$ (NoiseOSPF). NoiseOSPF is doing quite well, gaining a minimum of 40% over InvCapOSPF. Thus our weight settings are pretty robust to noise, and this implies that we only need a rough estimate of the demand matrix.

Note that the results obtained with D_2 with weights optimized for D_1 (TwinOSPF) are much worse than those with $\tilde{D}_1$ (NoiseOSPF), meaning weight settings are more sensible to structural changes than to noise.

5.5 Differentiated service

We now outline how our positive experience with multiple demand matrices could also be applied with differentiated service. Suppose we have two classes of customers: gold customers that were promised a guaranteed bandwidth, and normal customers. We want to ensure that gold customers packets are routed in the first 60% of the capacity of each link. Suppose we can find a weight setting such that we remain below 60% of the capacity if we only send gold customers demands. Then we can ensure with this weight setting that gold customers packets will be send below 60% of the capacity by giving them the priority over normal customers using weighted fair queuing. Let D_1 be the demand matrix describing the bandwidth promised to gold customers. We need to route D_1 with max-utilization $3/5$. Recall that our objective function Φ is designed to keep max-utilization below 1, which is also evident in our experiments with DirectOSPF. To satisfy our gold customers, keeping their max-utilization below $3/5$, we can therefore optimize with respect to $\Phi(G,(10/6D_1),w)$.

However, this approach does not take normal customers into account, and could lead to really bad situations for them. To remedy this, let D_2 be a demand matrix estimating the total traffic, including the gold customers. We want to provide good best-effort service with respect to D_1,

while keeping the objective of getting the gold customers max-utilization below $3/5$. This problem has been addressed for MPLS [21] where it is just a multi-commodity flow problem.

This leads us to a combined objective function of the form

$$\alpha\Phi(G,(10/6D_1),w)+\Phi(G,D_2,w)$$

where α is a parameter, that we can just optimize as we did for multiple demand matrices. If α is sufficiently high, we only care about the first term, and then we are pretty sure to stay below 60% capacity for gold customers, if at all possible. However, as soon as we have satisfied the gold customers, we better start worrying about best effort service for everybody as in the second term, so the natural optimization is to use the smallest α for which the gold customers get satisfied. This smallest α can be found by a binary search.

The interesting thing here is that our successful experiments with multiple demand matrices indicate that we can satisfy both gold customers and normal customers with OSPF/IS-IS without compromising either.

6 Conclusion

We have presented a system of algorithms for efficient OSPF traffic management in a changing world. As described in §5, we can optimize efficiently over a few peak demand matrices, say representing day and evening US traffic and international traffic in the night, and thereby produce a weight setting covering all demand matrices dominated by convex combinations of these peaks. The noise tolerance from §5 implies that we don't need to find the absolute peaks, as long as we get within a reasonable neighborhood of them. Further, the results from §4 indicate that our weight setting is good for new hot-spots, and most link failures, so generally, we do not expect to have to change the weight setting.

However, by simulation, we may discover that a few critical link failures can cause problems. For these few links we pre-compute a few weight changes to be applied in case they fail. Similar pro-active strategies can be applied against other predicted changes.

In case the whole structure of the demand matrix evolves to a degree that our weight setting is no longer satisfactory, we expect to reestablish good performance with just a few changes. Here the code from §3 can be used to generally monitor the network, keeping the network operator informed whether he can improve network performance with a few changes. The current running times of 1-2 hours may be considered too slow for such on-line traffic engineering. However, as is typical for local search, we can always make a less exhaustive search to terminate earlier. Within 10-15 minutes, one often get within 5% of the presented results. Also, there are much faster machines on the market. Anyhow, our system is expected to be most interesting to network operators in cases where real problems are experienced in the network, and where the alternative is to buy new hardware. In such a case, 1-2 hours of CPU time is cheap.

Besides the above concrete results, we believe that we have set a quite general framework for efficient traffic management, and many of the objectives could be reused within MPLS routing. Particularly this covers the convexity idea.

In combination, our techniques provide and efficient system for OSPF/IS-IS traffic management in a changing world. It is now an integrated part of AT&T's NetScope/Bravo traffic engineering tool for IP networks [14].

Acknowledgment

We would like to thank Jennifer Rexford for some very useful and patient comments to different drafts of this paper.

References

[1] E. H. L. Aarts and J. K. Lenstra, editors. *Local Search in Combinatorial Optimization.* Discrete Mathematics and Optimization. Wiley-Interscience, Chichester, England, 1997.

[2] W. Aiello, F. Chung, and L. Lu. A random model for massive graphs. In *Proc. 32nd ACM Symp. on Theory of Computing (STOC)*, pages 171–179, 2000.

[3] D. Awduche, A. Chiu, A. Elwalid, I. Widjaja, and X. Xiao. A framework for internet traffic engineering. Network Working Group, Internet Draft (work in progress), `http://search.ietf.org/internet-drafts/ draft-ietf-tewg-framework-02.txt`, 2000.

[4] D. O. Awduche. MPLS and traffic engineering in IP networks. *IEEE Communications Magazine*, 37(12):42–47, 1999.

[5] D. O. Awduche, J. Malcolm, J. Agogbua, M. O'Dell, and J. McManus. Requirements for traffic engineering over MPLS. Network Working Group, Request for Comments, `http://search.ietf.org/rfc/rfc2702.txt`, September 1999.

[6] Y. Bin, J. Cao, D. Davis, and S.V. Wiel. Time-varying network tomography: Router link data. In *Symposium on the Interface: Computing Science and Statistics, Schaumbur*, 1999.

[7] A. Bley, M. Grötchel, and R. Wessäly. Design of broadband virtual private networks: Model and heuristics for the B-WiN. Technical Report SC 98-13, Konrad-Zuse-Zentrum für Informationstechnik Berlin, 1998. To appear in *Proc. DIMACS Workshop on Robust Communication Networks and Survivability, AMS-DIMACS Series.*

[8] Caida. `http://www.caida.org/tools/visualization/mapnet/Data/`.

[9] R. Callon. Use of OSI IS-IS for routing in TCP/IP and dual environments. Network Working Group, Request for Comments: 1195, `http://search.ietf.org/rfc/rfc1195.txt`, December 1990.

[10] K. Calvert, M. Doar, and E. W. Zegura. Modeling Internet topology. *IEEE Communications Magazine*, 35(6):160–163, 1997.

[11] Cisco. Configuring OSPF, 1997. Documentation at `http://www.cisco.com/univerc/cc/td/doc/product/software/ios113ed/113ed_cr/np1_c/1cospf.htm`.

[12] N.G. Duffield and M. Grossglauser. Trajectory sampling for direct traffic observation. In *Proc. ACM SIGCOMM'00: Conf. Applications, Technologies, Architectures, and Protocols for Computer Communications*, pages 271–282, 2000.

[13] M. Faloutsos, P. Faloutsos, and C. Faloutsos. On power-law relationships of the internet topology. In *Proc. ACM SIGCOMM'99: Conf. Applications, Technologies, Architectures, and Protocols for Computer Communications*, pages 251–262, 1999.

[14] A. Feldmann, A. Greenberg, C. Lund, N. Reingold, and J. Rexford. NetScope: Traffic engineering for IP networks. *IEEE Network Magazine (Special Issue on Internet Traffic Engineering)*, 14(2):11–19, March/April 2000.

[15] A. Feldmann, A. Greenberg, C. Lund, N. Reingold, J. Rexford, and F. True. Deriving traffic demands for operational IP networks: Methodology and experience. In *Proc. ACM SIGCOMM'00: Conf. Applications, Technologies, Architectures, and Protocols for Computer Communications*, August/September 2000.

[16] A. Feldmann, J. Rexford, and R. Caceres. Efficient policies for carrying web traffic over flow-switched networks. *IEEE/ACM Transactions on Networking*, 6(6):673–685, 1998.

[17] B. Fortz and M. Thorup. Increasing internet capacity using local search. Technical Report IS-MG 2000/21, Université Libre de Bruxelles, 2000. `http://smg.ulb.ac.be/Preprints/Fortz00_21.html`. Submitted.

[18] B. Fortz and M. Thorup. Internet traffic engineering by optimizing OSPF weights. In *Proc. 19th IEEE Conf. on Computer Communications (INFOCOM)*, pages 519–528, 2000.

[19] F.Y.S. Lin and J.L. Wang. Minimax open shortest path first routing algorithms in networks supporing the SMDS services. In *Proc. IEEE International Conference on Communications (ICC)*, volume 2, pages 666–670, 1993.

[20] Lucent. Internet protocol network configurator, 2001. Tool described at `http://www.lucent.com/OS/ipnc.html`.

[21] D. Mitra and K.G.Ramakrishnan. A case study of multiservice, multipriority traffic engineering design for data networks. In *Proc. IEEE GLOBECOM*, pages 1077–1083, 1999.

[22] J. T. Moy. OSPF version 2. Network Working Group, Request for Comments: 1247, `http://search.ietf.org/rfc/rfc1247.txt`, July 1991.

[23] M.A. Rodrigues and K.G. Ramakrishnan. Optimal routing in data networks, 1994. Presentation at *International Telecommunications Symposium (ITS)*, Rio de Janeiro, Brazil.

[24] E. C. Rosen, A. Viswanathan, and R. Callon. Multiprotocol label switching architecture. Network Working Group, Request for Comments, `http://search.ietf.org/rfc/rfc3031.txt`, 2001.

[25] A. Sridharan, S. Bhattacharyya, C. Diot, R. Guerin, and J. Jetcheva. On the impact of aggregation on the performance of traffic aware routing. `http://www.seas.upenn.edu:8080/~ guerin/`. Submitted, 2000.

[26] P. L. Toint. Transportation modelling and operations research: A fruitful connection. In M. Labbé, G. Laporte, K. Tanczos, and P. L. Toint, editors, *Operations Research and Decision Aid Methodologies in Traffic and Transportation Management*, volume 166 of *NATO ASI series: Ser. F, Computer and systems sciences*, pages 1–27. Springer, 1998.

[27] B. M. Waxman. Routing of multipoint connections. *IEEE Jour. Selected Areas in Communications (Special Issue on Broadband Packet Communications)*, 6(9):1617–1622, 1988.

[28] E. W. Zegura. GT-ITM: Georgia tech internetwork topology models (software). `http://www.cc.gatech.edu/ fac/Ellen.Zegura/gt-itm/gt-itm.tar.gz`, 1996.

[29] E. W. Zegura, K. L. Calvert, and S. Bhattacharjee. How to model an internetwork. In *Proc. 15th IEEE Conf. on Computer Communications (INFOCOM)*, pages 594–602, 1996.