Robust optimization of OSPF/IS-IS weights

Bernard Fortz[1], Mikkel Thorup[2]
[1] I.A.G., Université Catholique de Louvain
Place des Doyens 1, B-1348 Louvain-la-Neuve, Belgium
Phone: +32-10 47 83 88, fax: +32-10 47 83 24
Email: fortz@poms.ucl.ac.be
[2] AT&T Labs-Research
Shannon Laboratory, Florham Park, NJ 07932, USA
Email:mthorup@research.att.com

Abstract

In this paper, we adapt the heuristic of Fortz and Thorup for optimizing the weights of Shortest Path First protocols such as Open Shortest Path First (OSPF) or Intermediate System-Intermediate System (IS-IS), in order to take into account failure scenarios.

More precisely, we want to find a set of weights that is robust to all single link failures. A direct application of the original heuristic, evaluating all the link failures, is too time consuming for realistic networks, so we developed a method based on a critical set of scenarios aimed to be representative of the whole set of scenarios. This allows us to make the problem manageable and achieve very robust solutions.

keywords Traffic Engineering, Taboo Search, Metaheuristics, IP Networks, Routing, Operation Research

1 Introduction

Shortest Path First (SPF) protocols such as Open Shortest Path First (OSPF) [11] or Intermediate System-Intermediate System (IS-IS) [4] 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 [5] 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. 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. The routing must be effective not only when the network is running without failure, but also in each case of single link failure.

In the context of a *fixed* network (i.e. with a fixed topology and no failure), with a fixed known demand matrix this problem has already been addressed experimentally in [9] 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 [2, 6, 3, 14].

However, as stipulated in [1], demand matrices and networks change. In [10], several scenarios of change are considered, with the objective of changing as few weights as possible. One of the results obtained shows the robustness of OSPF routing with optimized weights with respect to link failures. In this application, weights are optimized for the no-failure case, and a method to recover the performance with a few weight changes is provided. However, as described in [7], operators do not like to change weights, so it would be preferable to take into account the failure scenarios when optimizing the weights, in order to obtain a robust weight setting. This paper describes how the heuristic of Fortz and Thorup [8, 9] for the no-failure case can be adapted to take into account failure scenarios.

Our objective is to optimize network parameters with respect to a robustness objective function that requires the evaluation of all possible link failure scenarios. We would like to use our previous local search heuristic, but evaluating all link failures for each solution was intractable for the problems considered. We circumvented this problem by developing a set of critical link failures, which were the only ones considered in the inner loop of the local search. In an outer loop, we tested on all possible link failures the solution from the inner loop for the critical link failures. Based on this test, we sometimes added new link failures to the critical set. In the experiments reported, we managed to use critical sets that were two orders of magnitude smaller than the

set of all link failures, yet we managed to get very good with respect to all possible link failures. As a result, our critical set idea provided a speed-up by two orders of magnitude, allowing us to deal with the AT&T's IP backbone, which is the largest today. This critical set idea is of a very general nature and we expect it to find many other applications in the future.

A similar problem is considered in [13]. In this paper, only the maximum utilization in the normal state and for the worst link failure are optimized. The approach evaluates all link failures during the optimization, but the search space is reduced by considering only paths with a limited hop-count. Results are presented for a small network with 16 nodes and 68 arcs, while our approach allows to deal with much larger networks.

In Section 2, we recall the basic weight setting problem and the heuristic for the no-failure case. After describing why this heuristic cannot be used without modifications when all link failures are taken into account, we describe in Section 3 how it is adapted for this case. Preliminary numerical results are reported in Section 4.

2 The OSPF weight setting problem

In this section, we recall the basics of the OSPF weight setting problem and the heuristic proposed in [8, 9] for solving it.

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 directed network $G = (N,A)$ with a capacity c_a for each $a \in A$, and a demand matrix D that, for each pair $(s,t) \in N \times N$, $s \neq t$, tells the demand $D(s,t)$ in traffic flow between s and t. We sometimes refer to the non-zero entries of D as the *demands*. The set of arcs leaving a node u is denoted by $\delta^+(u) := \{(u,v) : (u,v) \in A\}$ while the set of arcs entering a node u is denoted by $\delta^-(u) := \{(v,u) : (v,u) \in A\}$

With each arc $a \in A$, we associate a cost function $\Phi_a(l_a)$ of the load l_a, depending on how close the load is to the capacity c_a. We assume in the following that Φ_a is an strictly increasing and convex function. Our formal objective is to distribute the demanded flow so as to minimize the sum

$$\Phi = \sum_{a \in A} \Phi_a(l_a)$$

of the resulting costs over all arcs. Usually, Φ_a increases rapidly as loads exceeds capacities, and our objective typically implies that we keep the max-utilization $\max_{a \in A} l_a/c_a$ below 1, or at least below 1.1, if at all possible.

In this general routing problem, there are no limitations to how we can distribute the flow between the paths. With each pair $(s,t) \in N \times N$ and each arc $a \in A$, we associate a variable $f_a^{(s,t)}$ telling how much of the traffic flow from s to t goes over a. Moreover, for each arc $a \in A$, variable l_a represents the total load on arc a, i.e. the sum of the flows going over a. With these notation, the problem can be formulated as a classical multi-commodity flow problem [8].

In our experiments, Φ_a are piecewise linear functions, with $\Phi_a(0) = 0$ and derivative

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

The objective function was chosen on the basis of discussions on costs with people close to the AT&T IP backbone. Motivations on our choice for the objective function and the different model assumptions are discussed in detail in [9], and, for a closely related application, in [10]. A description of the general infrastructure behind this kind of OSPF/IS-IS traffic engineering is given in [7].

2.2 The OSPF weight setting problem

The most commonly used intra-domain internet routing protocols today are shortest path protocols such as Open Shortest Path First (OSPF) [12]. OSPF does not support a free distribution of flow between source and destination as defined above in the general routing problem. In OSPF, the network operator assigns a weight w_a to each link $a \in A$, and shortest paths from each router to each destination are computed using these weights as lengths of the links. In practice, link weights are integer encoded on 16 bits, therefore they can take any value between 1 and 65,535. In each router, represented by a node of the graph, the next link on all shortest paths to all possible destinations is stored in a table. 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 splitting is done using pseudo-random methods leading to an approximately even splitting. For simplicity, we assume that the splitting is exactly even (for AT&T's WorldNet this simplification leads to reasonable estimates).

More precisely, given a set of weights $(w_a)_{a\in A}$, the length of a path is then the sum of its arc weights, and we have the extra condition that all flow leaving a node aimed at a given destination is evenly spread over the first arcs on shortest paths to that destination. Therefore, for each source-destination pair $(s,t) \in N \times N$ and for each arc $a \in \delta^+(u)$ for some node $u \in N$, we have that $f_a^{(s,t)} = 0$ if a is not on a shortest path from s to t, and that $f_a^{(s,t)} = f_{a'}^{(s,t)}$ if both $a \in \delta^+(u)$ and $a' \in \delta^+(u)$ are on shortest paths from s to t. Note that the routing of the demands is completely determined by the shortest paths which in turn are determined by the weights we assign to the arcs.

The quality of OSPF routing depends highly on the choice of weights. Nevertheless, as recommended by Cisco (a major router vendor) [5], these are often just set inversely proportional to the capacities of the links, without taking any knowledge of the demand into account.

The *OSPF weight setting problem* is to set the weights so as to minimize the cost of the resulting routing. This problem is NP-hard [8].

2.3 Local search heuristic

In OSPF routing, for each arc $a \in A$, we have to choose a weight w_a. These weights uniquely determine the shortest paths, the routing of traffic flow, the loads on the arcs, and finally, the value of the cost function Φ.

Suppose that we want to minimize a function f over a set X of feasible solutions. Local search techniques are iterative procedures that for each iteration define a neighborhood $\mathcal{N}(x) \subseteq X$ for the current solution $x \in X$, and then choose the next solution x' from this neighborhood. Often we want the neighbor $x' \in \mathcal{N}(x)$ to improve on f in the sense that $f(x') < f(x)$.

In the remainder of this section, we first describe the neighborhood structure we apply to solve the weight setting problem. Second, using hashing tables, we address the problem of avoiding cycling. These hashing tables are also used to avoid repetitions in the neighborhood exploration. While the neighborhood search aims at intensifying the search in a promising region, it is often of great practical importance to search a new region when the neighborhood search fails to improve the best solution for a while. These techniques are called search diversification. We refer the reader to [8] for a description of the diversification techniques we use.

A solution of the weight setting problem is completely characterized by its vector $w = (w_a)_{a\in A}$ of weights, where $w_a \in W$, the set of possible weights. We define a neighbor $w' \in \mathcal{N}(w)$ of w by one of the two following operations applied to w.

Single weight change. This simple modification consists in changing a single weight in w. We define a neighbor w' of w for each arc $a \in A$ and for each possible weight $t \in W \backslash \{w_a\}$ by setting $w'(a) = t$ and $w'(b) = w_b$ for all $b \neq a$.

Evenly balancing flows. Assuming that the cost function Φ_a for an arc $a \in A$ is increasing and convex, meaning that we want to avoid highly congested arcs, we want to split the flow as evenly as possible between different arcs.

More precisely, consider a demand node t such that $\sum_{s\in N} D(s,t) > 0$ and some part of the demand going to t goes through a given node u. Intuitively, we would like OSPF routing to split the flow to t going through u evenly along arcs leaving u. This is the case if every arc in $\delta^+(u)$ belongs to a shortest path from u to t. More precisely, if $\delta^+(u) = \{a_i : 1 \leq i \leq p\}$, and if P_i is one of the shortest paths from the tail of a_i to t, for $i = 1, \ldots, p$, then we want to set w' such that

$$w'_{a_i} + w'(P_i) = w'_{a_j} + w'(P_j) \qquad 1 \leq i, j \leq p,$$

where $w'(P_i)$ denotes the sum of the weights of the arcs belonging to P_i. A simple way of achieving this goal is to set

$$w'(a) = \begin{cases} w^* - w(P_i) & \text{if } a = a_i, \text{ for } i = 1, \ldots, p, \\ w_a & \text{otherwise.} \end{cases}$$

where $w^* = 1 + \max_{i=1,\ldots,p}\{w(P_i)\}$.

A drawback of this approach is that an arc that does not belong to one of the shortest paths from u to t may already be congested, and the modifications of weights we propose will send more flow on this congested arc, an obviously undesirable feature. We therefore decided to choose at random a threshold ratio θ between 0.25 and 1, and we only modify weights for arcs in the maximal subset B of $\delta^+(u)$ such that

$$w_{a_i} + w(P_i) \leq w_{a_j} + w(P_j) \;\; \forall i : a_i \in B, \; j : a_j \notin B,$$
$$l_a^w \leq \theta \, c_a \;\; \forall a \in B,$$

where l_a^w denotes the load on a resulting from weight vector w. The last relation implies that the utilization of an arc $a \in B$ resulting from the weight vector w is less than or equal to θ, so that we can avoid sending flow on already congested arcs. In this way, flow leaving u towards t can only change for arcs in B, and choosing θ at random allows to diversify the search.

This choice of B does not ensure that weights remain below w_{max}. This can be done by adding the condition $\max_{i:a_i\in B} w(P_i) - \min_{i:a_i\in B} w(P_i) \leq w_{max}$ when choosing B.

The simplest local search heuristic is the descent method that, at each iteration, selects the best element in the neighborhood and stops when this element does not improve the objective function. This approach leads to a local minimum that is often far from the optimal solution of the problem, and heuristics allowing non-improving moves have been considered. Unfortunately, non-improving moves can lead to cycling, and one must provide mechanisms to avoid it.

Our choice was to use hashing: hash functions compress solutions into single integer values, sending different solutions into the same integer with small probability. We use a boolean table T to record if a value produced by the hash function $h()$ has been encountered. At the beginning of the algorithm, all entries in T are set to false. If w is the solution produced at a given iteration, we set $T(h(w))$ to true, and, while searching the neighborhood, we reject any solution w' such that $T(h(w'))$ is true. Checking that a solution has been encountered is therefore performed in constant time. The hash function we used is described in [8].

3 Single link failures

The above heuristic has been designed to provide a weight setting for a single demand matrix and a fixed network. We now consider the possibility of link failures. We define a *state* of the network as a subset $S \subseteq A$ of arcs containing the arcs that are operational. The states considered in this paper are the *normal state* A and the *single link failure states* $A \setminus \{a\}$ for each link $a \in A$. In SPF protocols, 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. These shortest paths are updated each time the network state changes. Given a network state S, a weight setting w and a vector of arc capacities c, we denote by $\Phi(S,w,c)$ the cost of the routing obtained with weight setting w in state S of the network, using the piecewise linear cost function described in the previous section.

We suppose that once the operator has fixed the set of OSPF weights, he does not want to change it whatever the state of the network is. The possibility of getting a better routing in case of failures by allowing a few weight changes has been studied in [10].

The cost function we use has been designed in such a way that it tries to keep the flow on each link below the capacity of that link. This is an objective we want to maintain for each link failure. In the normal state, however, the operator usually wants the flows to remain much more below the capacity, in order to be more robust in cases of increasing demand and to ensure capacity will be available to perform the rerouting in case of failure. Let α be the maximal ratio of the capacity the network operator wants to use in the normal state. In our experiments, we assumed $\alpha = 0.6$. Therefore, in the normal state, the operator wants w to minimize $\Phi(A,w,\alpha c)$. We suppose here that the operator gives an equal importance to the quality of the routing in the normal state and to its robustness (i.e. the quality of the routing in all the single link failure states). Moreover, we assume all the link failures have the same importance, but it is trivial to extend our results by giving a weight to each link failure.

Putting it all together, we want to find a weight setting w^* that solves

$$\min_w \Psi(w) := \frac{1}{2}\left(\Phi(A,w,\alpha c) + \frac{1}{m}\sum_{a\in A}\Phi(A\setminus\{a\},w,c)\right) \tag{2}$$

where c is the capacity vector and $m := |A|$ the number of links in the network. A brute force approach would be to use the heuristic presented in the previous section to optimize $\Psi(w)$. However, this would require the evaluation of all the $m+1$ scenarios for each weight setting encountered, therefore increasing the computing time by a factor m (even with dynamic updates of shortest paths and flows as proposed in [8]).

Our approach to reduce the computing time is the following. We hope that only a few link failures will be representative of "bad cases" and will contribute for a large part of the total cost in (2). At each iteration, we maintain a list of *critical links* $\mathcal{C}$ and only evaluate the cost function restricted to the corresponding states, i.e. we evaluate each weight setting w in the neighborhood of the current iterate with the cost function

$$\Psi(w,\mathcal{C}) := \frac{1}{2}\left(\Phi(A,w,\alpha c) + \frac{1}{|\mathcal{C}|}\sum_{a\in\mathcal{C}}\Phi(A\setminus\{a\},w,c)\right).$$

The heuristic starts with $\mathcal{C} = \emptyset$. It is essentially the same as in the previous section, adapted to optimize $\Psi(w,\mathcal{C})$, with the addition that $\mathcal{C}$ is updated every T iterations. We update $\mathcal{C}$ as follows. Let $u(a,w)$ be the maximum utilization with weight setting w when arc a has failed, i.e.

$$u(a,w) = \max_{b\in A\setminus\{a\}} \frac{l(b,w,a)}{c_b},$$

where $l(b,w,a)$ denotes the load on arc b with weight setting w when a has failed, and let $\overline{u}$ be the average maximum utilization over all scenarios in the critical set, i.e. $\overline{u} := \frac{1}{|\mathcal{C}|}\sum_{a\in\mathcal{C}} u(a,w)$. We first choose the arc a not in the critical set that maximizes $u(a,w)$. If $u(a,w) > \overline{u}$, we add a to $\mathcal{C}$. Moreover, we want to keep $\mathcal{C}$ of small size. To this end, we fix a maximal size K and we remove the arc that minimizes $u(a,w)$ over $\mathcal{C}$ from the critical set each time its size exceeds K.

	InvCap					FT					Robust-FT				
		Normal		Worst			Normal		Worst			Normal		Worst	
Network	Ψ	Φ	u	Φ	u	Ψ	Φ	u	Φ	u	Ψ	Φ	u	Φ	u
HIER-50-148-2488	1700	3384	1.31	303.5	1.61	0.457	0.630	0.67	11.98	1.35	0.186	0.248	0.64	0.184	1.04
HIER-100-280-1934	16.81	33.45	0.87	10.41	1.34	0.779	0.174	0.64	1.384	1.24	0.124	0.136	0.56	1.233	1.23
RAND-100-403-29813	17.56	34.98	0.86	0.464	1.16	0.321	0.530	0.67	0.608	1.20	0.178	0.283	0.57	0.127	0.90

Table 1: Numerical results

4 Preliminary numerical results

In Table 1, we present some results obtained with the heuristic described in this paper. These are preliminary results and a fine tuning of the parameters of the heuristic as well as more extensive testing are under way. Three networks coming from [8] are used. Two of them are 2-level graphs (HIER-) and one pure random graph (RAND-). We also performed some experiments on AT&T's IP backbone, but these could not be reproduced here for proprietary reasons. However, results obtained were similar to those presented here.

The methodology used to build the graphs and the demand matrices is widely discussed in [8, 10]. The numbers after the type of graph denote the number of nodes, the number of arcs and the total demand. The level of demand was chosen such that the maximum utilization obtained with the original heuristic for optimizing the weights is close to the critical level α for the normal state. In the table, InvCap denotes Cisco's recommended strategy of setting the weight of an arc inversely proportional to its capacity, FT denotes the original weight setting heuristic of [8, 9] and Robust-FT is the heuristic presented in this paper. We performed 500 iterations of Robust-FT, with $\alpha = 0.6$, $T = 10$ and $K = 5$. The starting point for the optimization was InvCap. For each heuristic, Ψ is the total cost (2), and Φ and u are the cost and the maximum utilization, given for the normal state and the worst link failure. From these results, it clearly appears that both FT and Robust-FT outperform InvCap. Moreover, Robust-FT does pretty well in improving both the total cost and the worst failure case, as well as the normal state. Remark that FT was applied to optimize Φ without scaling the capacities, while Robust-FT tries harder to push the maximum utilization below α. This explains why Robust-FT improves on the normal state, even if one could have expected a deterioration due to the simultaneous optimization of failure cases.

The computing times for 500 iterations of Robust-FT were of the same magnitude as performing 5000 iterations of FT. As Robust-FT outperforms FT while ensuring a more robust solution in cases of failure, we think it is a powerful tool for traffic engineering in IP networks.

References

[1] D. Awduche, A. Chiu, A. Elwalid, I. Widjaja, and X. Xiaro. 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.

[2] W. Ben-Ameur, N. Michel, E. Gourdin, and B. Liau. Routing strategies for IP networks. *Telektronikk*, 2/3:145–158, 2001.

[3] A. Bley, M. Grötchel, and R. Wessäly. Design of broadband virtual private networks: Model and heuristics for the B-WiN. In *Proc. DIMACS Workshop on Robust Communication Networks and Survivability, AMS-DIMACS Series 53*, pages 1–16, 1998.

[4] 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.

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

[6] M. Ericsson, M.G.C Resende, and P.M. Pardalos. A genetic algorithm for the weight setting problem in OSPF routing, 2001. To appear in *J. Combinatorial Optimization* in 2002.

[7] B. Fortz, J. Rexford, and M. Thorup. Traffic engineering with traditional IP routing protocols. *IEEE Communications Magazine*, 40(10):118–124, 2002.

[8] 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`.

[9] 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.

[10] B. Fortz and M. Thorup. Optimizing OSPF/IS-IS weights in a changing world. *IEEE Journal on Selected Areas in Communications*, 20(4):756–767, 2002.

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

[12] J. T. Moy. *OSPF: Anatomy of an Internet Routing Protocal*. Addison-Wesley, 1999.

[13] A. Nucci, B. Schroeder, S. Bhattacharyya, N. Taft, and C. Diot. IGP link weight assignment for transient link failures. to appear in ITC18, Berlin (Germany), 31 August - 5 September 2003.

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