Internet Traffic Engineering by Optimizing OSPF Weights

Bernard Fortz
Service de Mathématiques de la Gestion
Institut de Statistique et de Recherche Opérationnelle
Université Libre de Bruxelles
Brussels, Belgium
bfortz@smg.ulb.ac.be

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

Abstract—**Open Shortest Path First (OSPF) is the most commonly used intra-domain internet routing protocol. Traffic flow is routed along shortest paths, splitting flow at nodes where several outgoing links are on shortest paths to the destination. The weights of the links, and thereby the shortest path routes, can be changed by the network operator. The weights could be set proportional to their physical distances, but often the main goal is to avoid congestion, i.e. overloading of links, and the standard heuristic recommended by Cisco is to make the weight of a link inversely proportional to its capacity.**

Our starting point was a proposed AT&T WorldNet backbone with demands projected from previous measurements. The desire was to optimize the weight setting based on the projected demands. We showed that optimizing the weight settings for a given set of demands is NP-hard, so we resorted to a local search heuristic. Surprisingly it turned out that for the proposed AT&T WorldNet backbone, we found weight settings that performed within a few percent from that of the *optimal general routing* **where the flow for each demand is optimally distributed over all paths between source and destination. This contrasts the common belief that OSPF routing leads to congestion and it shows that for the network and demand matrix studied we cannot get a substantially better load balancing by switching to the proposed more flexible Multi-protocol Label Switching (MPLS) technologies.**

Our techniques were also tested on synthetic internetworks, based on a model of Zegura et al. (INFOCOM'96), for which we did not always get quite as close to the optimal general routing. However, we compared with standard heuristics, such as weights inversely proportional to the capacity or proportional to the physical distances, and found that, for the same network and capacities, we could support a 50%–110% increase in the demands.

Our assumed demand matrix can also be seen as modeling service level agreements (SLAs) with customers, with demands representing guarantees of throughput for virtual leased lines.

Keywords—**OSPF, MPLS, traffic engineering, local search, hashing tables, dynamic shortest paths, multi-commodity network flows.**

I. Introduction

PROVISIONING an Internet Service Provider (ISP) backbone network for intra-domain IP traffic is a big challenge, particularly due to rapid growth of the network and user demands. At times, the network topology and capacity may seem insufficient to meet the current demands. At the same time, there is mounting pressure for ISPs to provide Quality of Service (QoS) in terms of Service Level Agreements (SLAs) with customers, with loose guarantees on delay, loss, and throughput. All of these issues point to the importance of *traffic engineering*, making more efficient use of existing network resources by tailoring routes to the prevailing traffic.

A. The general routing problem

The *general routing problem* is defined as follows. Our network is a directed graph, or multi-graph, $G = (N, A)$ whose nodes and arcs represent routers and the links between them. Each arc a has a capacity $c(a)$ which is a measure for the amount of traffic flow it can take. In addition to the capacitated network, we are given a demand matrix D that for each pair (s,t) of nodes tells us how much traffic flow we will need to send from s to t. We shall 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. The routing problem is now, for each non-zero demand $D(s,t)$, to distribute the demanded flow over paths from s to t. Here, in the general routing problem, we assume there are no limitations to how we can distribute the flow between the paths from s to t.

The above definition of the general routing problem is equivalent to the one used e.g. in Awduche et al. [1]. Its most controversial feature is the assumption that we have an estimate of a demand matrix. This demand matrix could, as in our case for the proposed AT&T WorldNet backbone, be based on concrete measures of the flow between source-destination pairs. The demand matrix could also be based on a concrete set of customer subscriptions to virtual leased line services. Our demand matrix assumption does not accommodate unpredicted bursts in traffic. However, we can deal with more predictable periodic changes, say between morning and evening, simply by thinking of it as two independent routing problems: one for the morning, and one for the evening.

Having decided on a routing, the load $\ell(a)$ on an arc a is the total flow over a, that is $\ell(a)$ is the sum over all demands of the amount of flow for that demand which is sent over a. The utilization of a link a is $\ell(a)/c(a)$.

Loosely speaking, our objective is to keep the loads within the capacities. More precisely, our cost function Φ sums the cost of the arcs, and the cost of an arc a has to do with the relation between $\ell(a)$ and $c(a)$. In our experimental study, we had

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

where for all $a \in A$, $\Phi_a(0) = 0$ and

$$\phi'_a(x) = \begin{cases} 1 & \text{for} \quad 0 \leq x/c(a) < 1/3 \\ 3 & \text{for} \quad 1/3 \leq x/c(a) < 2/3 \\ 10 & \text{for} \quad 2/3 \leq x/c(a) < 9/10 \\ 70 & \text{for} \quad 9/10 \leq x/c(a) < 1 \\ 500 & \text{for} \quad 1 \leq x/c(a) < 11/10 \\ 5000 & \text{for} \quad 11/10 \leq x/c(a) < \infty \end{cases}$$

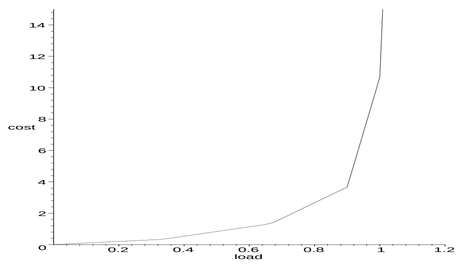

Fig. 1. Arc cost $\Phi_a(\ell(a))$ as a function of load $\ell(a)$ for arc capacity $c(a) = 1$.

The function Φ_a is illustrated in Fig. 2. The idea behind Φ_a is that it is cheap to send flow over an arc with a small utilization. As the utilization approaches 100%, it becomes more expensive, for example because we get more sensitive to bursts. If the utilization goes above 100%, we get heavily penalized, and when the utilization goes above 110% the penalty gets so high that this should never happen.

The exact definition of the objective function is not so important for our techniques, as long as it is a a piece-wise linear increasing and convex function.

Proposition 1: If each Φ_a is a piece-wise linear increasing and convex function, we can solve the general routing problem optimally in polynomial time.

Knowing the optimal solution for the general routing problem is an important benchmark for judging the quality of solutions based on, say, OSPF routing.

The above objective function provides a general best effort measure. We have previously claimed that our approach can also be used if the demand matrix is modeling service level agreements (SLAs) with customers, with demands representing guarantees of throughput for virtual leased lines. In this case, we are more interested in ensuring that no packet gets sent across overloaded arcs, so our objective is to minimize a maximum rather than a sum over the arcs. However, due to the very high penalty for overloaded arcs, our objective function favours solutions without overloaded arcs. A general advantage to working with a sum rather than a maximum is that even if there is a bottleneck link that is forced to be heavily loaded, our objective function still cares about minimizing the loads in the rest of the network.

B. OSPF versus MPLS routing protocols

Unfortunately, most intra-domain internet routing protocols today do not support a free distribution of flow between source and destination as defined above in the general routing problem. The most common protocol today is Open Shortest Path First (OSPF) [2]. In this protocol, 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 demand going in the router is sent to its destination by splitting the flow between the links that are on the shortest paths to the destination. The exact mechanics of the splitting can be somewhat complicated, depending on the implementation. Here, as a simplifying approximation, we assume that it is an even split.

The quality of OSPF routing depends highly on the choice of weights. Nevertheless, as recommended by Cisco [3], these are often just set inversely proportional to the capacities of the links, without taking any knowledge of the demand into account.

It is widely believed that the OSPF protocol is not flexible enough to give good load balancing as defined, for example, in our objective function. This is one of the reasons for introducing the more flexible Multi-protocol Label Switching (MPLS) technologies ([1], [4]). With MPLS one can in principle decide the path for each individual packet. Hence, we can simulate a solution to the general routing problem by distributing the packets on the paths between a source-destination pair using the same distribution as we used for the flow. The MPLS technology has some disadvantages. First of all, MPLS is not yet widely deployed, let alone tested. Second OSPF routing is simpler in the sense that the routing is completely determined by one weight for each arc. That is, we do not need to make individual routing decisions for each source/destination pair. Also, if a link fails, the weights on the remaining links immediately determines the new routing.

C. Our results

The general question studied in this paper is: *Can a sufficiently clever weight settings make OSPF routing perform nearly as well as optimal general/MPLS routing?*

Our first answer is negative: for arbitrary n, we construct an instance of the routing problem on $\approx n^3$ nodes where *any* OSPF routing has its average flow on arcs with utilization $\Omega(n)$ times higher than the max-utilization in an optimal general solution. With our concrete objective function, this demonstrates a gap of a factor approaching 5000 between the cost of the optimal general routing and the cost of the optimal OSPF routing.

The next natural question is: how well does OSPF routing perform on real networks. In particular we wanted to answer this question for a proposed AT&T WorldNet backbone. In addition, we studied synthetic internetworks, generated as suggested by Calvert, Bhattacharjee, Daor, and Zegura [5], [6]. Finding a perfect answer is hard in the sense that it is NP-hard to find an optimal setting of the OSPF weights for an arbitrary network. Instead we resorted to a local search heuristic, not guaranteed to find optimal solutions. Very surprisingly, it turned out that for the proposed AT&T WorldNet backbone, the heuristic found weight settings making OSPF routing performing within a few percent from the optimal general routing. Thus for the proposed AT&T WorldNet backbone with our projected demands, and with our concrete objective function, there would be no substantial traffic engineering gain in switching from the existing well-tested and understood robust OSPF technology to the new MPLS alternative.

For the synthetic networks, we did not always get quite as close to the optimal general routing. However, we compared our local search heuristic with standard heuristics, such as weights inversely proportional to the capacities or proportional to the physical distances, and found that, for the same network and capacities, we could support a 50%–110% increase in the demands, both with respect to our concrete cost function and, simultaneously, and with respect to keeping the max-utilization below 100%.

D. Technical contributions

Our local search heuristic is original in its use of hash tables to avoid cycling and for search diversification. A first attempt of using hashing tables to avoid cycling in local search was made by Woodruff and Zemel [7], in conjunction with tabu search. Our approach goes further and avoids completely the problem specific definitions of solution attributes and tabu mechanisms, leading to an algorithm that is conceptually simpler, easier to implement, and to adapt to other problems.

Our local search heuristic is also original in its use of more advanced dynamic graph algorithms. Computing the OSPF routing resulting from a given setting of the weights turned out to be the computational bottleneck of our local search algorithm, as many different solutions are evaluated during a neighborhood exploration. However, our neighborhood structure allows only a few local changes in the weights. We therefore developed efficient algorithms to update the routing and recompute the cost of a solution when a few weights are changed. These speed-ups are critical for the local search to reach a good solution within reasonable time bounds.

E. Contents

In Section II we formalize our general routing model as a linear program, thereby proving Proposition 1. We present in Section III a scaled cost function that will allow us to compare costs across different sizes and topologies of networks. In Section IV, a family of networks is constructed, demonstrating a large gap between OSPF and multi-commodity flow routing. In Section V we present our local search algorithm, guiding the search with hash tables. In Section VI we show how to speed-up the calculations using dynamic graph algorithms. In Section VII, we report the numerical experiments. Finally, in Section VIII we discuss our findings.

Because of space limitations, we defer to the journal version the proof that it is NP-hard to find an optimal weight setting for OSPF routing. In the journal version, based on collaborative work with Johnson and Papadimitriou, we will even prove it NP-hard to find a weight setting getting within a factor 1.77 from optimality for arbitrary graphs.

II. Model

A. Optimal routing

Recall that we are given a directed network $G = (N, A)$ with a capacity $c(a)$ for each $a \in A$. Furthermore, we have a demand matrix D that for each pair $(s,t) \in N \times N$ tells the demand $D(s,t)$ in traffic flow between s and t. We will sometimes refer to the non-zero entries of D as the *demands*. With each pair (s,t) and each arc a, we associate a variable $f_a^{(s,t)}$ telling how much of the traffic flow from s to t goes over a. Variable $\ell(a)$ represents the total load on arc a, i.e. the sum of the flows going over a, and Φ_a is used to model the piece-wise linear cost function of arc a.

With this notation, the general routing problem can be formulated as the following linear program.

$$\min \Phi = \sum_{a \in A} \Phi_a$$

subject to

$$\sum_{x:(x,y)\in A} f_{(x,y)}^{(s,t)} - \sum_{z:(y,z)\in A} f_{(y,z)}^{(s,t)} = \begin{cases} -D(s,t) & \text{if } y=s, \\ D(s,t) & \text{if } y=t, \\ 0 & \text{otherwise,} \end{cases} \qquad y,s,t \in N, \quad (1)$$

$$\ell(a) = \sum_{(s,t)\in N\times N} f_a^{(s,t)} \qquad a \in A, \quad (2)$$

$$\Phi_a \geq \ell(a) \qquad a \in A, \quad (3)$$

$$\Phi_a \geq 3\ell(a) - \tfrac{2}{3}c(a) \qquad a \in A, \quad (4)$$

$$\Phi_a \geq 10\ell(a) - \tfrac{16}{3}c(a) \qquad a \in A, \quad (5)$$

$$\Phi_a \geq 70\ell(a) - \tfrac{178}{3}c(a) \qquad a \in A, \quad (6)$$

$$\Phi_a \geq 500\ell(a) - \tfrac{1468}{3}c(a) \qquad a \in A, \quad (7)$$

$$\Phi_a \geq 5000\ell(a) - \tfrac{19468}{3}c(a) \qquad a \in A, \quad (8)$$

$$f_a^{(s,t)} \geq 0 \qquad a \in A;\ s,t \in N. \quad (9)$$

Constraints (1) are flow conservation constraints that ensure the desired traffic flow is routed from s to t, constraints (2) define the load on each arc and constraints (3) to (8) define the cost on each arc.

The above program is a complete linear programming formulation of the general routing problem, and hence it can be solved optimally in polynomial time (Khachiyan [8]), thus settling Proposition 1. In our experiments, we solved the above problems by calling CPLEX via AMPL. We shall use Φ_{OPT} to denote the optimal general routing cost.

B. OSPF routing

In OSPF routing, we choose a weight $w(a)$ for each arc. 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 a destination is evenly spread over arcs on shortest paths to that destination. More precisely, for each source-destination pair $(s,t) \in N \times N$ and for each node x, we have that $f_{(x,y)}^{(s,t)} = 0$ if (x,y) is not on a shortest path from s to t, and that $f_{(x,y)}^{(s,t)} = f_{(x,y')}^{(s,t)}$ if both (x,y) and (x,y') 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. Unfortunately, the above condition of splitting between shortest paths based on variable weights cannot be formulated as a linear program, and this extra condition makes the problem NP-hard.

We shall use $\Phi_{OptOSPF}$ to denote the optimal cost with OSPF routing.

III. Normalizing cost

We now introduce a normalizing scaling factor for the cost function that allows us to compare costs across different sizes and topologies of networks. To define the measure, we introduce

$$\Phi_{Uncap} = \sum_{(s,t)\in N\times N} (D(s,t) \cdot \mathrm{dist}_1(s,t)). \quad (10)$$

Above $\mathrm{dist}_1(\cdot)$ is distance measured with unit weights (hop count). Also, we let $\Phi_{UnitOSPF}$ denote the cost of OSPF routing with unit weights. Below we present several nice properties of Φ_{Uncap} and $\Phi_{UnitOSPF}$. It is (ii) that has inspired the name "Uncapacitated".

Lemma 2:
(i) Φ_{Uncap} is the total load if all traffic flow goes along unit weight shortest paths.
(ii) $\Phi_{Uncap} = \Phi_{UnitOSPF}$ if all arcs have unlimited capacity.
(iii) Φ_{Uncap} is the minimal total load of the network.
(iv) $\Phi_{Uncap} \leq \Phi_{OPT}$.
(v) $\Phi_{UnitOSPF} < 5000 \cdot \Phi_{Uncap}$.
Above, 5000 is the maximal value of Φ'_a.

Proof: Above (i) follows directly from the definition. Now (ii) follows from (i) since the ratio of cost over load on an arc is 1 if the capacity is more than 3 times the load. Further (iii) follows from (i) because sending flow along longer paths only increases the load. From (iii) we get (iv) since 1 is the smallest possible ratio of cost over load. Finally we get (v) from (ii) since decreasing the capacity of an arc with a given load increases the arc cost with strictly less than a factor 5000 if the capacity stays positive. ■

Our scaled cost is now defined as:

$$\Phi^* = \Phi / \Phi_{Uncap}. \tag{11}$$

From Lemma 2 (iv) and (v), we immediately get:

$$1 \leq \Phi^*_{OPT} \leq \Phi^*_{OptOSPF} \leq \Phi^*_{UnitOSPF} < 5000. \tag{12}$$

Note that if we get $\Phi^* = 1$, it means that we are routing along unit weight shortest paths with all loads staying below 1/3 of the capacity. In this ideal situation there is no point in increasing the capacities of the network.

A packet level view

Perhaps the most instructive way of understanding our cost functions is on the packet level. We can interpret the demand $D(s,t)$ as measuring how many packets — of identical sizes — we expect to send from s to t within a certain time frame. Each packet will follow a single path from s to t, and it pays a cost for each arc it uses. With our original cost function Φ, the per packet cost for using an arc is $\Phi_a(\ell(a))/\ell(a)$. Now, Φ can be obtained by summing the path cost for each packet.

The per packet arc cost $\Phi_a(\ell(a))/\ell(a)$ is a function ϕ of the *utilization* $\ell(a)/c(a)$. The function ϕ is depicted in Fig. 2. First ϕ is 1 while the utilization is $\leq 1/3$. Then ϕ increases to $10 - 16/3 = 10\frac{2}{3}$ for a full arc, and after that it grows rapidly towards 5000. The *cost factor* of a packet is the ratio of its current cost over the cost it would have paid if it could follow the unit-weight shortest path without hitting any utilization above 1/3, hence paying the minimal cost of 1 for each arc traversed. The latter ideal situation is what is measured by Φ_{Uncap}, and therefore Φ^* measures the weighted average packet cost factor where each packet is weighted proportionally to the unit-weight distance between its end points.

If a packet follows a shortest path, and if all arcs are exactly full, the cost factor is exactly $10\frac{2}{3}$. The same cost factor can of course be obtained by some arcs going above capacity and others going below, or by the packet following a longer detour using less congested arcs. Nevertheless, it is natural to say that a routing *congests* a network if $\Phi^* \geq 10\frac{2}{3}$.

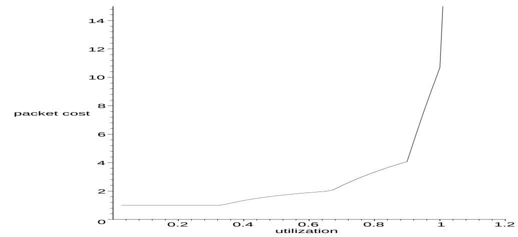

Fig. 2. Arc packet cost as function of utilization

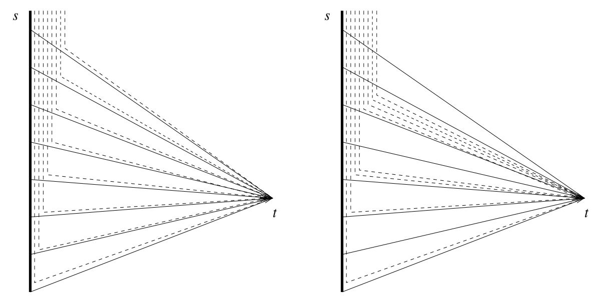

Fig. 3. Gap between general flow solution on the left and OSPF solution on the right

IV. Maximal gap between OPT and OSPF

From (12), it follows that the maximal gap between optimal general routing and optimal OSPF routing is less than a factor 5000. In Lemma 3, we will now show that the gap can in fact approach 5000. Our proof is based on a construction where OSPF leads to very bad congestion for any natural definition of congestion. More precisely, for arbitrary n, we construct an instances of the routing problem on $\approx n^3$ nodes with only one demand, and where all paths from the source to the destination are shortest with respect to unit weights, or hop count. In this instance, *any* OSPF routing has its average flow on arcs with utilization $\Omega(n)$ times higher than the max-utilization in an optimal general solution. With our concrete objective function, this demonstrates a gap of a factor approaching 5000 between the cost of the optimal general routing and the cost of the optimal OSPF routing.

Lemma 3: There is a family of networks G_n so that the optimal general routing approaches being 5000 times better than the optimal OSPF routing as $n \to \infty$.

Proof: In our construction, we have only one demand with source s and destination t, that is, $D(x,y) > 0$ if and only if $(x,y) = (s,t)$. The demand is n. Starting from s, we have a directed path $P = (s, x_1, ..., x_n)$ where each arc has capacity $3n$. For each i, we have a path Q_i of length $n^2 - i$ from x_i to t. Thus, all paths from s to t have length n^2. Each arc in each Q_i has capacity 3. The graph is illustrated in Fig. 3 with P being the thick high capacity path going down on the left side.

In the general routing model, we send the flow down P, letting 1 unit branch out each Q_i. This means that no arc gets more flow than a third of its capacity, so the optimal cost Φ^n_{OPT}

satisfies

$$\Phi^n_{OPT} \leq n(n^2+n) = (1+o(1))n^3.$$

In the OSPF model, we can freely decide which Q_i we will use, but because of the even splitting, the first Q_i used will get half the flow, i.e. $n/2$ units, the second will get $n/4$ units, etc. Asymptotically this means that almost all the flow will go along arcs with load a factor $\Omega(n)$ above their capacity, and since all paths uses at least $n^2 - n$ arcs in some Q_i, the OSPF cost $\Phi^n_{OptOSPF}$ satisfies

$$\Phi^n_{OptOSPF} \geq (1-o(1))5000n^3.$$

We conclude that the ratio of the OSPF cost over the optimal cost is such that $\frac{\Phi^n_{OptOSPF}}{\Phi^n_{OPT}} \geq (1-o(1))5000 \rightarrow 5000$ as $n \rightarrow \infty$. ■

V. OSPF Weight Setting using Local Search

A. Overview of the 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 cost function Φ. In this section, we present a local search heuristic to determine a weights vector $(w_a)_{a\in A}$ that minimizes Φ. We let $W := \{1, \dots, w_{max}\}$ denote the set of possible weights.

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

Differences between local search heuristics arise essentially from the definition of the neighborhood, the way it is explored, and the choice of the next solution from the neighborhood. Descent methods consider the entire neighborhood, select an improving neighbor and stop when a local minimum is found. Meta-heuristics such as tabu search or simulated annealing allow non-improving moves while applying restrictions to the neighborhood to avoid cycling. An extensive survey of local search and its application can be found in Aarts and Lenstra [9].

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 and are addressed at the end of the section.

B. Neighborhood structure

A solution of the weight setting problem is completely characterized by its vector w of 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 $\Phi_a()$ 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 x. Intuitively, we would like OSPF routing to split the flow to t going through x evenly along arcs leaving x. This is the case if all the arcs leaving x belong to a shortest path from x to t. More precisely, if $x_1, x_2, \dots, x_p$ are the nodes adjacent to x, and if P_i is one of the shortest paths from x_i to t, for $i = 1, \dots, p$, as illustrated in Fig. 4, then we want to set w' such that

$$\begin{aligned} w'_{(x,x_1)} + w'(P_1) &= w'_{(x,x_2)} + w'(P_2) \\ &= \dots = w'_{(x,x_p)} + w'(P_p). \end{aligned}$$

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

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

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

A drawback of this approach is that an arc that does not belong to one of the shortest paths from x 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 arcs leaving x such that

$$\begin{aligned} w_{(x,x_i)} + w(P_i) \leq w_{(x,x_j)} + w(P_j) \;\; &\forall i \in B, \; j \notin B, \\ l_{(x,x_i)}(w) \leq \theta \; c(x, x_i) \;\; &\forall i \in B. \end{aligned}$$

In this way, flow going from n to t can only change for arcs in B, and choosing θ at random allows to diversify the search. Another drawback of this approach is that it does not ensure that weights remain below w_{max}. This can be done by adding the condition that $\max_{i\in B} w(P_i) - \min_{i\in B} w(P_i) \leq w_{max}$ is satisfied when choosing B.

C. Guiding the search with hashing tables

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. Tabu search algorithms (Glover [10]), for example, make use of a tabu list that records some attributes of solutions encountered during the recent iterations and forbids any solution having the same attributes.

We developed a search strategy that completely avoids cycling without the need to store complex solution attributes. The

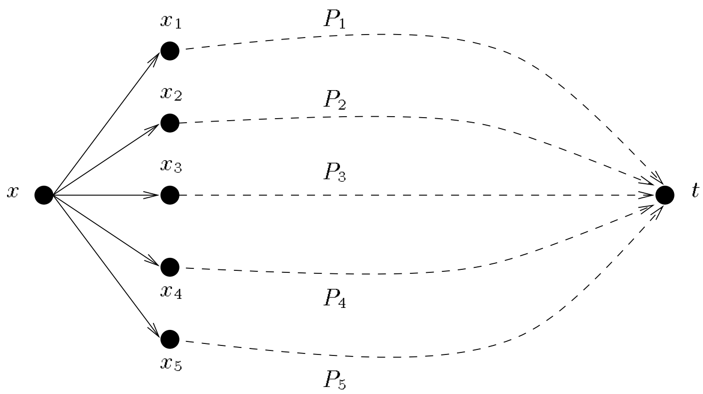

Fig. 4. The second type of move tries to make all paths form n to t of equal length.

solutions to our problem are $|A|$-dimensional integer vectors. Our approach maps these vectors to integers, by means of a hashing function h, chosen as described in [11]. Let l be the number of bits used to represent these integers. We use a boolean table T to record if a value produced by the hashing function has been encountered. As we need an entry in T for each possible value returned by h, the size of T is 2^l. In our implementation, $l = 16$. 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.

This approach completely eliminates cycling, but may also reject an excellent solution having the same hashing value as a solution met before. However, if h is chosen carefully, the probability of collision becomes negligible. A first attempt of using hashing tables to avoid cycling in local search was made by Woodruff and Zemel [7], in conjunction with tabu search. It differs from our approach since we completely eliminate the tabu lists and the definition of solution attributes, and we store the values for all the solutions encountered, while Woodruff and Zemel only record recent iterations (as in tabu search again). Moreover, they store the hash values encountered in a list, leading to a time linear in the number of stored values to check if a solution must be rejected, while with our boolean table, this is done in constant time.

D. Speeding up neighborhood evaluation

Due to our complex neighborhood structure, it turned out that several moves often lead to the same weight settings. For efficiency, we would like to avoid evaluation of these equivalent moves. Again, hashing tables are a useful tool to achieve this goal : inside a neighborhood exploration, we define a secondary hashing table used to store the encountered weight settings as above, and we do not evaluate moves leading to a hashing value already met.

The neighborhood structure we use has also the drawback that the number of neighbors of a given solution is very large, and exploring the neighborhood completely may be too time consuming. To avoid this drawback, we only evaluate a randomly selected set of neighbors.

We start by evaluating 20 % of the neighborhood. Each time the current solution is improved, we divide the size of the sampling by 3, while we multiply it by 10 each time the current solution is not improved. Moreover, we enforce sampling at least 1 % of the neighborhood.

E. Diversification

Another important ingredient for local search efficiency is diversification. The aim of diversification is to escape from regions that have been explored for a while without any improvement, and to search regions as yet unexplored.

In our particular case, many weight settings can lead to the same routing. Therefore, we observed that when a local minimum is reached, it has many neighbors having the same cost, leading to long series of iterations with the same cost value. To escape from these "long valleys" of the search space, the secondary hashing table is again used. This table is generally reset at the end of each iteration, since we want to avoid repetitions inside a single iteration only. However, if the neighborhood exploration does not lead to a solution better than the current one, we do not reset the table. If this happens for several iterations, more and more collisions will occur and more potentially good solutions will be excluded, forcing the algorithm to escape from the region currently explored. For these collisions to appear at a reasonable rate, the size of the secondary hashing table must be small compared to the primary one. In our experiments, its size is 20 times the number of arcs in the network.

This approach for diversification is useful to avoid regions with a lot of local minima with the same cost, but is not sufficient to completely escape from one region and go to a possibly more attractive one. Therefore, each time the best solution found is not improved for 300 iterations, we randomly perturb the current solution in order to explore a new region from the search space. The perturbation consists of adding a randomly selected perturbation, uniformly chosen between -2 and +2, to 10 % of the weights.

VI. Cost Evaluation

We will now first show how to evaluate our cost function for the static case of a network with a specified weight setting. Computing this cost function from scratch is unfortunately too time consuming for our local search, so afterwards, we will show how to reuse computations, exploiting that there are only few weight changes between a current solution and any solution in its neighborhood.

A. The static case

We are given a directed multigraph $G = (N, A)$ with arc capacities $\{c_a\}_{a \in A}$, demand matrix D, and weights $\{w_a\}_{a \in A}$. For the instances considered, the graph is sparse with $|A| = O(|N|)$. Moreover, in the weighted graph the maximal distance between any two nodes is $O(|N|)$.

We want to compute our cost function Φ. The basic problem is to compute the loads resulting from the weight setting. We will consider one destination t at a time, and compute the total flow from all sources $s \in N$ to t. This gives rise to a certain partial load $l_a^t = \sum_{s \in N} f_a^{(s,t)}$ for each arc. Having done the above computation for each destination t, we can compute the load l_a on arc a as $\sum_{t \in N} l_a^t$.

To compute the flow to t, our first step is to use Dijkstra's algorithm to compute all distances to t (normally Dijkstra's algorithm computes the distances away from some source, but we can just apply such an implementation of Dijkstra's algorithm to the graph obtained by reversing the orientation of all arcs in G). Having computed the distance d_x^t to t for each node, we compute the set A^t of arcs on shortest paths to t, that is,

$$A^t = \{(x,y) \in A : d_x^t - d_y^t = w_{(x,y)}\}$$

For each node x, let δ_x^t denote its outdegree in A^t, i.e. $\delta_x^t = |\{y \in N : (x,y) \in A^t\}|$.

Observation 4: For all $(y,z) \in A^t$,

$$l_{(y,z)}^t = \frac{1}{\delta_y^t}(D(y,t) + \sum_{(x,y)\in A^t} l_{(x,y)}^t).$$

Using Observation 4, we can now compute all the loads $l_{(y,z)}^t$ as follows. The nodes $y \in N$ are visited in order of decreasing distance d_y^t to t. When visiting a node y, we first set $l = \frac{1}{\delta_y^t}(D(y,t) + \sum_{(x,y)\in A^t} l_{(x,y)}^t)$. Second we set $l_{(y,z)}^t = l$ for each $(y,z) \in A^t$.

To see that the above algorithm works correctly, we note the invariant that when we start visiting a node y, we have correct loads l_a^t on all arcs a leaving nodes coming before y. In particular this implies that all the arcs (x,y) entering y have correctly computed loads $l_{(x,y)}^t$, and hence when visiting y, we compute the correct load $l_{(y,z)}^t$ for arcs (y,z) leaving y.

Using bucketing for the priority queue in Dijkstra's algorithm, the computation for each destination takes $O(|A|) = O(|N|)$ time, and hence our total time bound is $O(|N|^2)$.

B. The dynamic case

In our local search we want to evaluate the cost of many different weight settings, and these evaluations are a bottleneck for our computation. To save time, we will try to exploit the fact that when we evaluate consecutive weight settings, typically only a few arc weights change. Thus it makes sense to try to be lazy and not recompute everything from scratch, but to reuse as much as possible. With respect to shortest paths, this idea is already well studied (Ramalingam and Reps [12]), and we can apply their algorithm directly. Their basic result is that, for the recomputation, we only spend time proportional to the number of arcs incident to nodes x whose distance d_x^t to t changes. In our experiments there were typically only very few changes, so the gain was substantial - in the order of factor 20 for a 100 node graph. Similar positive experiences with this laziness have been reported in Frigioni et al. [13].

The set of changed distances immediately gives us a set of "update" arcs to be added to or deleted from A^t. We will now present a lazy method for finding the changes of loads. We will operate with a set M of "critical" nodes. Initially, M consists of all nodes with an incoming or outgoing update arc. We repeat the following until M is empty: First, we take the node $y \in M$ which maximizes the updated distance d_y^t and remove y from M. Second, set $l = \frac{1}{\delta_y^t}(D(y,t) + \sum_{(x,y)\in A^t} l_{(x,y)}^t)$. Finally, for each $(y,z) \in A^t$, where A^t is also updated, if $l \neq l_{(y,z)}^t$, set $l_{(y,z)}^t = l$ and add z to M.

To see that the above suffices, first note that the nodes visited are considered in order of decreasing distances. This follows because we always take the node at the maximal distance and because when we add a new node z to M, it is closer to t than the currently visited node y. Consequently, our dynamic algorithm behaves exactly as our static algorithm except that it does not treat nodes not in M. However, all nodes whose incoming or outgoing arc set changes, or whose incoming arc loads change are put in M, so if a node is skipped, we know that the loads around it would be exactly the same as in the previous evaluation.

VII. Numerical Experiments

We present here our results obtained with a proposed AT&T WorldNet backbone as well as synthetic internetworks.

Besides comparing our local search heuristic (HeurOSPF) with the general optimum (OPT), we compared it with OSPF routing with "oblivious" weight settings based on properties of the arc alone but ignoring the rest of the network. The oblivious heuristics are InvCapOSPF setting the weight of an arc inversely proportional to its capacity as recommended by Cisco [3], UnitOSPF just setting all arc weights to 1, L2OSPF setting the weight proportional to its physical Euclidean distance (L_2 norm), and RandomOSPF, just choosing the weights randomly.

Our local search heuristic starts with randomly generated weights and performs 5000 iterations, which for the largest graphs took approximately one hour. The random starting point was weights chosen for RandomOSPF, so the initial cost of our local search is that of RandomOSPF.

The results for the AT&T WorldNet backbone with different scalings of the projected demand matrix are presented in Table I. In each entry we have the normalized cost Φ^* introduced in Section III. The normalized cost is followed by the max-utilization in parenthesis. For all the OSPF schemes, the normalized cost and max-utilization are calculated for the same weight setting and routing. However, for OPT, the optimal normalized cost and the optimal max-utilization are computed independently with different routing. We do not expect any general routing to be able to get the optimal normalized cost and max-utilization simultaneously. The results are also depicted graphically in Figure 5. The first graph shows the normalized cost and the horizontal line shows our threshold of $10\frac{2}{3}$ for regarding the network as congested. The second graph shows the max-utilization.

The synthetic internetworks were produced using the generator GT-ITM [14], based on a model of Calvert, Bhattacharjee, Daor, and Zegura [5], [6]. 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. The above model does not include a model for the demands. 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

$$\alpha O_x D_y C_{(x,y)} e^{-\delta(x,y)/2\Delta}$$

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 modelling 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. The results for the synthetic networks are presented in Figures 6–9.

VIII. Discussion

First consider the results for the AT&T WorldNet backbone with projected (non-scaled) demands in Table 5(*). As mentioned in the introduction, our heuristic, HeurOSPF, is within 2% from optimality. In contrast, the oblivious methods are all off by at least 15%.

Considering the general picture for the normalized costs in Figures 5–9, we see that L2OSPF and RandomOSPF compete to be worst. Then comes InvCapOSPF and UnitOSPF closely together, with InvCapOSPF being slightly better in all the figures but Figure 6. Recall that InvCapOSPF is Cisco's recommendation [3], so it is comforting to see that it is the better of the oblivious heuristics. The clear winner of the OSPF schemes is our HeurOSPF, which is, in fact, much closer to the general optimum than to the oblivous OSPF schemes.

To quantify the difference between the different schemes, note that all curves, except those for RandomOSPF, start off pretty flat, and then, quite suddenly, start increasing rapidly. This pattern is somewhat similar to that in Fig. 1. This is not surprising since Fig. 1 shows the curve for a network consisting of a single arc. The reason why RandomOSPF does not follow this pattern is that the weight settings are generated randomly for each entry. The jumps of the curve for Random in Figure 8 nicely illustrate the impact of luck in the weight setting. Interestingly, for a particular demand, the value of RandomOSPF is the value of the initial solution for our local search heuristic. However, the jumps of RandomOSPF are not transferred to HeurOSPF which hence seems oblivious to the quality of the initial solution.

Disregarding RandomOSPF, the most interesting comparison between the different schemes is the amount of demand they can cope with before the network gets too congested. In Section III, we defined $10\frac{2}{3}$ as the threshold for congestion, but the exact threshold is inconsequential. In our experiments, we see that HeurOSPF allows us to cope with 50%-110% more demand. Also, in all but Figure 7, HeurOSPF is less than 2% from being able to cope with the same demands as the optimal general routing OPT. In Figure 7, HeurOSPF is about 20% from OPT. Recall that it is NP-hard even to approximate the optimal cost of an OSPF solution, and we have no idea whether there exisits OSPF solutions closer to OPT than the ones found by our heuristic.

If we now turn out attention to the max-utilization, we get the same ordering of the schemes, with InvCapOSPF the winner among the oblivious schemes and HeurOSPF the overall best OSPF scheme. The step-like pattern of HeurOSPF show the impact of the changes in Φ'_a. For example, in Figure 5, we see how HeurOSPF fights to keep the max utilization below 1, in order to avoid the high penalty for getting load above capacity. Following the pattern in our analysis for the normalized cost, we can ask how much more demand we can deal with before getting max-utiliztion above 1, and again we see that HeurOSPF beats the oblivious schemes with at least 50%.

The fact that our HeurOSPF provides weight settings and routings that are simultaneously good both for our best effort type average cost function, and for the performance guarantee type measure of max-utilization indicates that the weights obtained are "universally good" and not just tuned for our particular cost function. Recall that the values for OPT are not for the same routings, and there may be no general routing getting simultaneously closer to the optimal cost and the optimal max-utilization than HeurOSPF. Anyhow, our HeurOSPF is generally so close to OPT that there is only limited scope for improvement.

In conclusion, our results indicate that in the context of known demands, a clever weight setting algorithm for OSPF routing is a powerful tool for increasing a network's ability to honor increasing demands, and that OSPF with clever weight setting can provide large parts of the potential gains of traffic engineering for supporting demands, even when compared with the possibilities of the much more flexible MPLS schemes.

Acknowledgment We would like to thank David Johnson and Jennifer Rexford for some very useful comments. The first author was sponsored by the AT&T Research Prize 1997.

References

[1] 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.

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

[3] Cisco, *Configuring OSPF*, 1997.

[4] E. C. Rosen, A. Viswanathan, and R. Callon, "Multiprotocol label switching architecture," Network Working Group, Internet Draft (work in progress), `http://search.ietf.org/internet-drafts/draft-ietf-mpls-arch-06.txt`, 1999.

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

[6] K. Calvert, M. Doar, and E. W. Zegura, "Modeling internet topology," *IEEE Communications Magazine*, vol. 35, no. 6, pp. 160–163, June 1997.

[7] D. L. Woodruff and E. Zemel, "Hashing vectors for tabu search," *Annals of Operations Research*, vol. 41, pp. 123–137, 1993.

[8] L. G. Khachiyan, "A polynomial time algorithm for linear programming," *Dokl. Akad. Nauk SSSR*, vol. 244, pp. 1093–1096, 1979.

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

[10] F. Glover, "Future paths for integer programming and links to artificial intelligence," *Computers & Operations Research*, vol. 13, pp. 533–549, 1986.

[11] M. Dietzfelbibger, "Universal hashing and k-wise independent random variables via integer arithmetic without primes," in *Proc. 13th Symp. on Theoretical Aspects of Computer Science (STACS), LNCS 1046*. 1996, pp. 22–24, Springer.

[12] G. Ramalingam and T. Reps, "An incremental algorithm for a generalization of the shortest-path problem," *Joanal of Algorithms*, vol. 21, no. 2, pp. 267–305, 1996.

[13] D. Frigioni, M. Ioffreda, U. Nanni, and G. Pasqualone, "Experimental analysis of dynamic algorithms for the single-source shortest path problem," *ACM Jounal of Experimental Algorithmics*, vol. 3, article 5, 1998.

[14] 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`.

Demand	InvCapOSPF	UnitOSPF	L2OSPF	RandomOSPF	HeurOSPF	OPT
3709	1.01 (0.15)	1.00 (0.15)	1.13 (0.23)	1.12 (0.35)	1.00 (0.17)	1.00 (0.10)
7417	1.01 (0.30)	1.00 (0.31)	1.15 (0.46)	1.91 (1.05)	1.00 (0.30)	1.00 (0.19)
11126	1.05 (0.45)	1.03 (0.46)	1.21 (0.70)	1.36 (0.66)	1.01 (0.34)	1.01 (0.29)
14835	1.15 (0.60)	1.13 (0.62)	1.42 (0.93)	12.76 (1.15)	1.05 (0.47)	1.04 (0.39)
(*) 18465	1.33 (0.75)	1.31 (0.77)	5.47 (1.16)	59.48 (1.32)	1.16 (0.59)	1.14 (0.48)
22252	1.62 (0.90)	1.59 (0.92)	44.90 (1.39)	86.54 (1.72)	1.32 (0.67)	1.30 (0.58)
25961	2.70 (1.05)	3.09 (1.08)	82.93 (1.62)	178.26 (1.86)	1.49 (0.78)	1.46 (0.68)
29670	17.61 (1.20)	21.78 (1.23)	113.22 (1.86)	207.86 (4.36)	1.67 (0.89)	1.63 (0.77)
33378	55.27 (1.35)	51.15 (1.39)	149.62 (2.09)	406.29 (1.93)	1.98 (1.00)	1.89 (0.87)
37087	106.93 (1.51)	93.85 (1.54)	222.56 (2.32)	476.57 (2.65)	2.44 (1.00)	2.33 (0.97)
40796	175.44 (1.66)	157.00 (1.69)	294.52 (2.55)	658.68 (3.09)	4.08 (1.10)	3.64 (1.06)
44504	246.22 (1.81)	228.30 (1.85)	370.79 (2.78)	715.52 (3.37)	15.86 (1.34)	13.06 (1.16)

TABLE I

AT&T'S PROPOSED BACKBONE WITH 90 NODES AND 274 ARCS AND SCALED PROJECTED DEMANDS, WITH (*) MARKING THE ORIGINAL UNSCALED DEMAND.

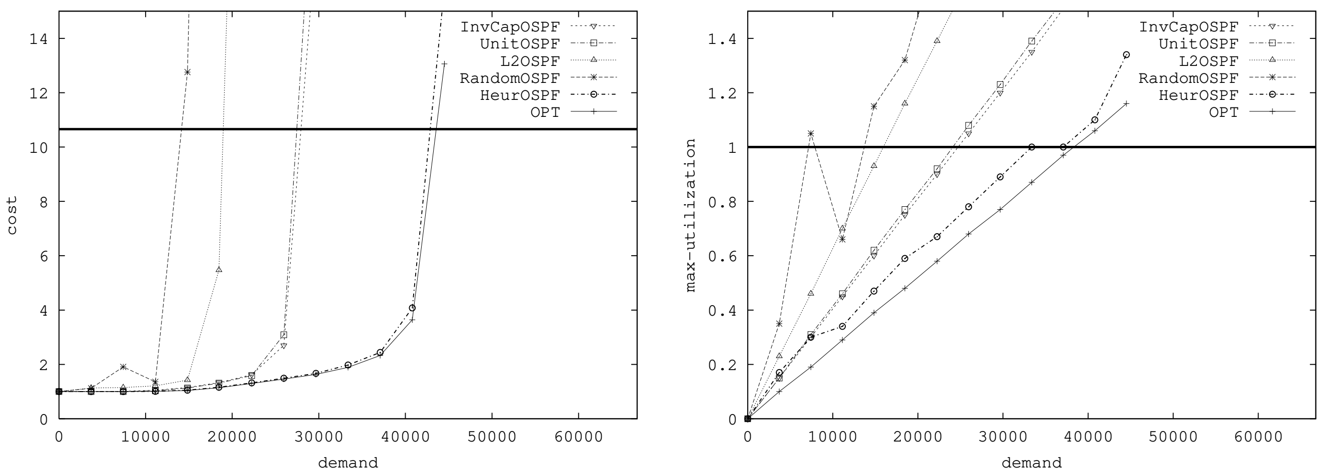

Fig. 5. AT&T's proposed backbone with 90 nodes and 274 arcs and scaled projected demands.

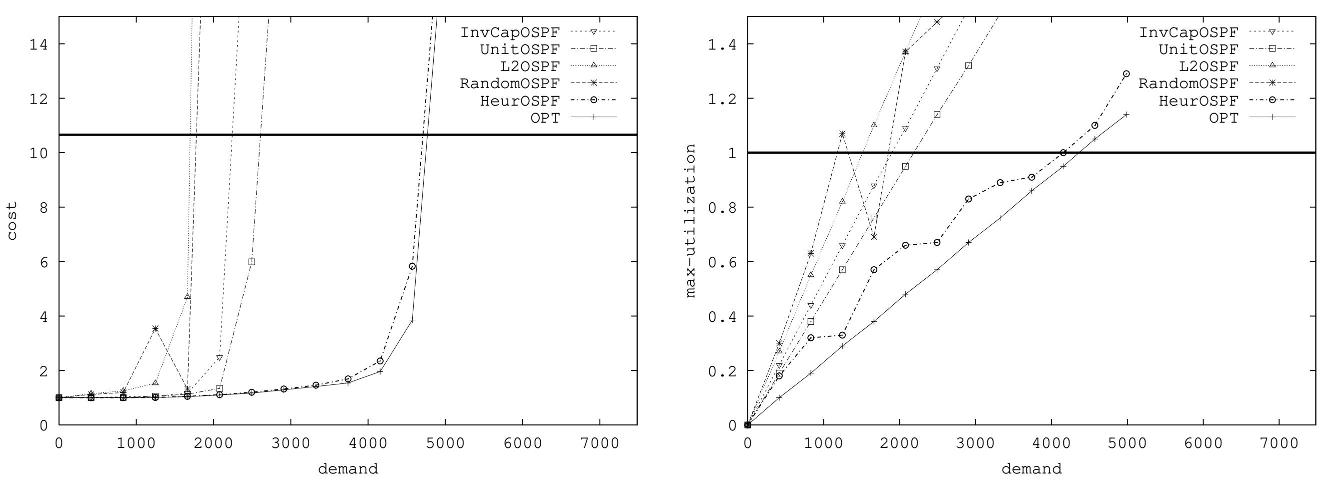

Fig. 6. 2-level graph with 50 nodes and 148 arcs.

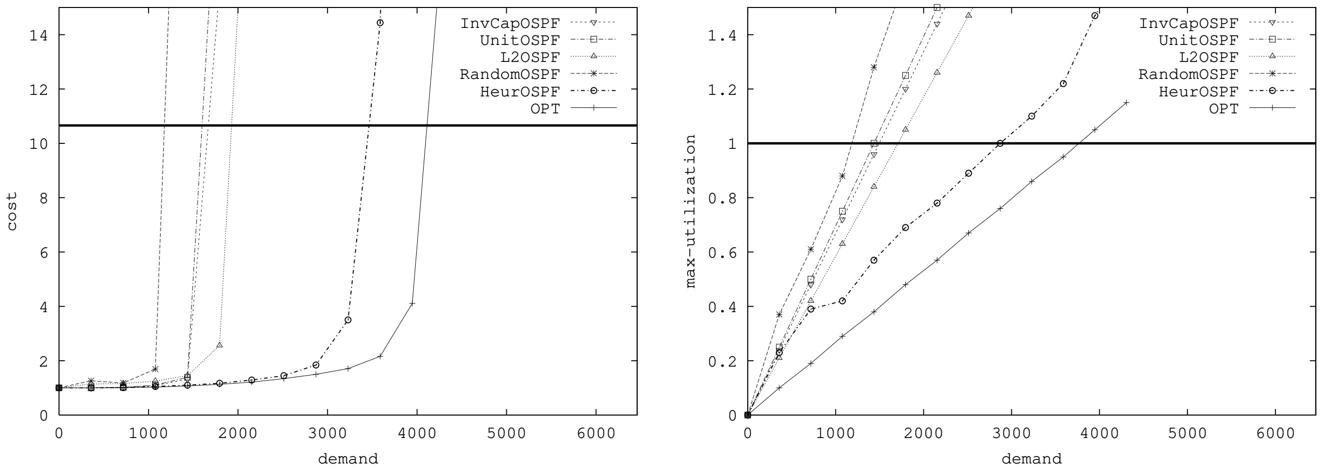

Fig. 7. 2-level graph with 50 nodes and 212 arcs.

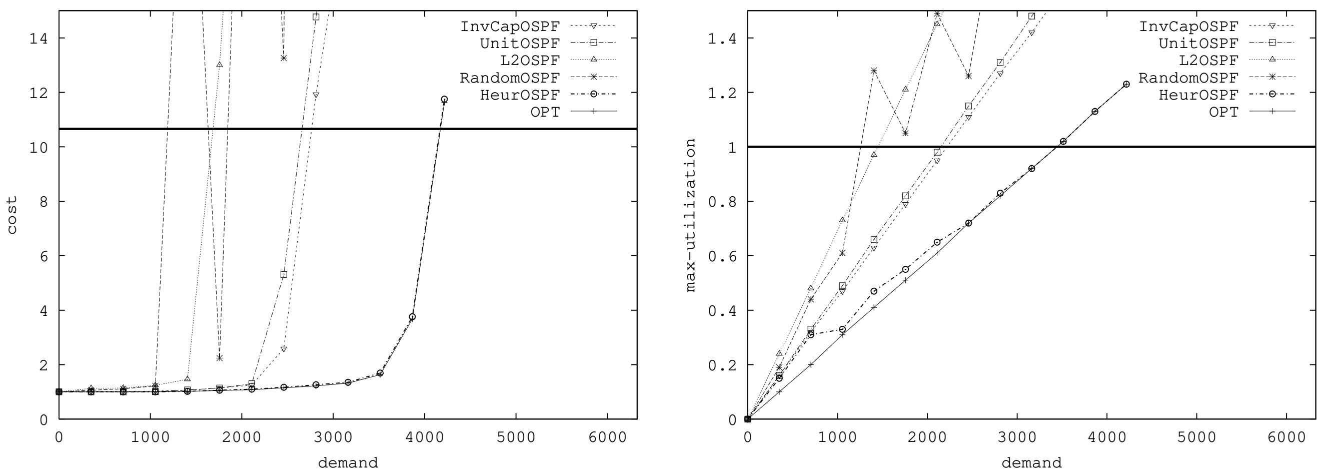

Fig. 8. 2-level graph with 100 nodes and 280 arcs.

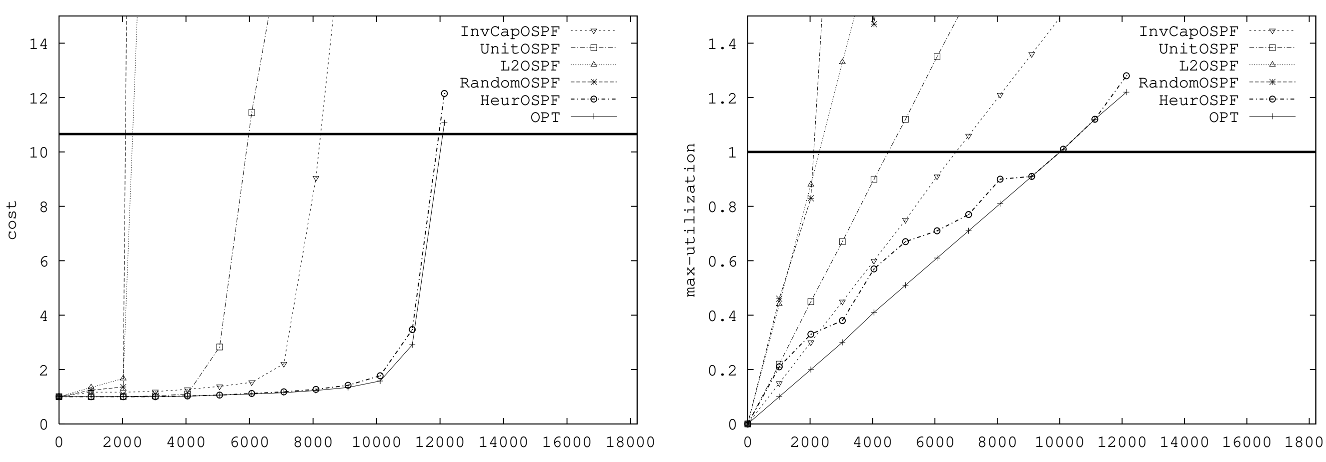

Fig. 9. 2-level graph with 100 nodes and 360 arcs.