A memetic algorithm for the weight setting problem in DEFT

Roger Reis[*], Marcus Ritt[*], Luciana S. Buriol[*], Mauricio G.C. Resende[†]

Abstract

To use interior gateway protocols like OSPF (Open Shortest Path First, [6]), IS-IS (Intermediate System-Intermediate System), and DEFT (Distributed Exponentially-weighted Flow Splitting, [7]) it is necessary to set the link weights to allow data routing. The problem of finding suitable weights for these protocols is known as the weight setting problem. Examples of objective functions are network congestion, link utilization, and latency. In the literature we can find a few papers describing methods to set the link weights according to some protocol. In this paper we present a new memetic algorithm which can be applied to both the OSPF and DEFT protocols, study its performance, and discuss future directions of this work.

1 Introduction

The Internet is divided into many Autonomous Systems (ASes). Each one controls its interior routing by an interior gateway protocol (IGP). Common interior gateway protocols like open shortest path first (OSPF) or intermediate system-intermediate system (IS-IS) allow the operator to tune the routes by setting integer weights on the network links. The problem of finding weights which optimize some objective function, for example total network congestion, link utilization, or latency, is called the *weight setting problem*.

To optimize the use of all network capacity, an objective function and a set of constraints are introduced to model the network. The problem of weight setting for OSPF is shown to be NP-Hard in [4]. Among the best results achieved for OSPF we refer to the tabu search approach in [4] and the memetic algorithm (MA) in [1]. In the following, we focus on the protocols OSPF and DEFT and show how the advances in computing weights for OSPF compares to DEFT, both using a MA approach.

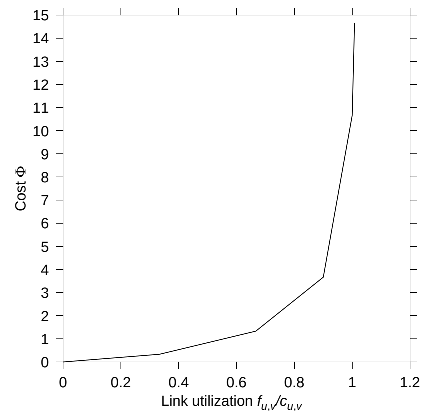

Figure 1: Link cost Φ depending on the link utilization for $c_{u,v} = 1$.

Let $G = (V, E)$ be a directed graph with link capacities $c_{u,v}$, and D a demand matrix where D_{ij} denotes the traffic demand from source node i to destination node j, for $1 \leq i, j \leq |V|$. We define T as the subset of nodes $T \subseteq V$ that comprises all the nodes that are destination of at least one demand pair, i.e.

$$T = \{v | D_{uv} > 0\}.$$

The routing problem is to find flows $f_{u,v}$ which satisfy all demands and minimize the total link utilization

$$\text{minimize} \qquad \sum_{(u,v)\in E} \Phi(f_{u,v}, c_{u,v}) \qquad (1)$$

where Φ is a link cost function. A typical choice for Φ is the piece-wise linear function shown in Figure 1 [4, 5].

Let $f^t_{u,v}$ be the flow on link (u, v) destined to node t. Then any resulting flow must respect the constraints of flow conservation at intermediate nodes $v \neq t$

$$\sum_{(u,v)\in E} f^t_{u,v} - \sum_{(v,w)\in E} f^t_{v,w} = D_{v,t} \qquad (2)$$

[*]Instituto de Informática, Universidade Federal do Rio Grande do Sul {rsreis,mrpritt,buriol}@inf.ufrgs.br

[†]Internet and Network Systems Research Center, AT&T Labs Research mgcr@research.att.com

and the individual flow aggregation

$$f_{u,v} = \sum_{t \in T} f^t_{u,v}. \tag{3}$$

Since the objective function and all constraints are linear, we can find an optimal solution by solving the linear program OPT given by Eqs. 1, 2 and 3 together with

$$0 \leq f^t_{u,v}, 0 \leq f_{u,v}. \tag{4}$$

OPT has no routing restrictions and is not employed in practice, but serves as a lower bound for practical routing protocols.

The paper is organized as follows. In Section 2 we introduce OSPF and DEFT, the two protocols that are of interest in this paper. Next, in Section 3, we present the MA proposed for the OSPF. In Section 4 we introduce the modifications applied in the MA presented in the previous section to allow it to implement DEFT rules. The results of the proposed algorithm are summarized in Section 5. Finally, Section 6 presents the main conclusions of this study, as well as future investigations on it.

2 OSPF and DEFT Protocols

In OSPF the flow is determined using integer weights $w_{u,v} \in [0, 2^{16} - 1]$ on each link. The routers exchange information about the links, including their weights. Each router u uses these weights to compute the shortest paths to all destinations. It then distributes incoming traffic destined to a node t equally amongst all outgoing links on shortest paths having t as destination.

DEFT relaxes this constraint. It allows real weights $w_{u,v} \in \mathbb{R}$ and distributes the flow amongst all outgoing links whose next node is closer to the destination. Links which are not part of a shortest path receive a flow which decreases with exponential penalties for longer paths lengths. Formally, let d^t_i be the distance from node i to destination t, and let $h^t_{u,v} = d^t_v + w_{u,v} - d^t_u$ be the distance gap of using the link (u, v) compared to the shortest path. Then, the non-normalized traffic fraction Γ for link (u, v), directed to t, is calculated as

$$\Gamma(h^t_{u,v}) = \begin{cases} e^{-h^t_{u,v}} & \text{if } d^t_u > d^t_v \\ 0 & \text{otherwise} \end{cases} \tag{5}$$

and the fraction of the total flow $\Gamma(h^t_{u,v}) / \sum_{v:(u,v)\in E} \Gamma(h^t_{u,v})$ is calculated for each outgoing link (u, v) of u. According to [7], in terms of total link cost and maximum utilization, there always exists a weight setting such that DEFT is better than OSPF.

Finding such weights, on the other hand, i.e. solving the weight setting problem optimally for these protocols is difficult. For example, finding the weights minimizing link utilization in OSPF is NP-hard [4]. Therefore, for OSPF, several authors have proposed heuristics solutions, including genetic algorithms [3], memetic algorithms [1], and tabu search [4, 5]. For DEFT, [7] proposed a two-stage iterative method, based on non-linear, non-smooth optimization.

3 A memetic algorithm for OSPF

In this section we describe the memetic algorithm previously proposed in the literature to solve the weight setting problem for OSPF. More details can be found in [1].

A memetic algorithm, also known as hybrid genetic algorithm, is a genetic algorithm augmented with a local search procedure to speed up the search by improving candidate solutions locally. In this context, a solution is called an *individual*, each element of the solution is a *gene*, a set of individuals is called a *population*, and each iteration of the algorithm is called a *generation*. It is a populational method in which, during each iteration, individuals are combined through a crossover procedure for generating new individuals that will form the next generation. The algorithm runs for a number of generations, aiming to improve the quality of solutions from one generation to the next. Each solution is evaluated by an objective function that, in this problem, is to minimize the network congestion.

In [3], a genetic algorithm is presented with structured population for OSPF routing. In this structure, individuals are classified in three classes, according to their fitness. Class A is composed of the best 25% of the individuals, class $\mathcal{C}$ is composed by the 5% less profitable solutions, and the remaining of the population composes class $\mathcal{B}$. The solutions from class A pass directly to the next generation. The solutions from class $\mathcal{C}$ are replaced by new ones randomly generated. The remaining solutions are replaced by solutions generated by the crossover procedure between a parent solution from class $\mathcal{A}$ and another from set $\mathcal{B} \cup \mathcal{C}$.

The crossover operator is a random key scheme that prioritizes (given 70% of chances) genes originated from parents in class $\mathcal{A}$. A local search approach is applied on each solution generated by a crossover operator. The local improvement procedure examines the effect of increasing the weights of a subset of the arcs. These candidate arcs are selected among those with the highest routing costs and whose weight is smaller than the maximum allowed. To reduce the routing cost of a candidate arc, the procedure attempts to increase its weight to induce a reduction on its load. If this leads to a reduction in the overall routing cost, the change is accepted, and the procedure is restarted. This procedure executes consecutive solution evaluations, that are expensive computational operations in this problem. To speed up this process, given a weight change, the shortest path graphs, as well as the flow allocation, are only updated, instead of recomputed from scratch.

The solution evaluation is the most expensive operation in terms of computational time. Given a set of integer weights, a shortest path graph G^t is computed, as well as the routing (flow allocation), for each destination node $t \in T$.

Next we discuss the modifications needed to consider the DEFT protocol.

4 A memetic algorithm applying DEFT rules

In this section we present the above mentioned memetic algorithm modified for applying DEFT rules.

We maintained the whole memetic algorithm structure and operators described in the previous section, only changing the evaluation procedure.

Recall that, following OSPF rules, the flow on each node is evenly split among all shortest path links outgoing this node with destination in t. In DEFT, the load in a node u is split among *all* outgoing links (u, v) (and not only the shortest path links) and that approaches t, i.e., $d_u^t > d_v^t$. Moreover, the load split is not equal among all links as it is in OSPF. DEFT applies an exponential penalty to longer paths between origin-destination pairs nodes such that more load is routed through links that lead to shorter paths.

Figure 2 describes in pseudocode how DEFT rules are implemented. As in OSPF, for each destination node $t \in T$ we compute the reverse shortest path graph G^t (lines 2-3). In lines 4-23 we detail the procedure `ComputePartialLoads` that implements the DEFT rules that allows flow be routed on non-shortest paths.

```
**procedure** $CostEvalDEFT(w = w_1, w_2, \ldots, w_{|V|})$
1    **for** $\forall\, t \in T$ ;
2        $d$ =ReverseDijkstra(t,w)
3        $G^t$ = ComputeShortestPathGraph(w,d)
4        [ComputePartialLoads($d$, $G^t$, $D$)]
5            H = sorted nodes in decreasing order of distances
6            **for** each $u \in H$
7                $\Gamma_{total}$ = 0
8                **for** each $v \in OUT(u)$
9                    **if** $d_u^t > d_v^t$ **then**
10                       $h_{u,v}^t = d_v^t + w_{u,v} - d_u^t$
11                       $\Gamma_{total}$ += $e^{-h_{u,v}^t}$
12                   **endif**
13               **endfor**
14               f = $(D_{u,v} + \sum_{(u,v)\in G^t} f_{(u,v)}^t)$ / $\Gamma_{total}$
15               **for** each $a \in OUT(u)$
16                   **if** $d_u^t > d_v^t$ **then**
17                       $h_{u,v}^t = d_v^t + w_{u,v} - d_u^t$
18                       $\gamma = e^{-h_{u,v}^t}$
19                       $f_{u,v}^t = f * \gamma$
20                   **endif**
21               **endfor**
22           **endfor**
23       [end of ComputePartialLoads]
24       for each $(u,v) \in A$ $f_{u,v} += f_{u,v}^t$
25   **endfor**
26   $\Phi = \sum_{(u,v)\in E} \phi(u,v)$
**end** CostEvalDEFT.
```

Figure 2: Pseudocode describing solution evaluation for DEFT

In line 5 we sort the nodes in decreasing order of their distances to t. Nodes are analyzed one by one, from the more distant to the closest. The loop in lines 8-13 calculates the sum (Γ_{total}) of the exponential function (Equation 5) for each outgoing link of the current node. We denote by $OUT(u) = \{v | (u, v) \in E\}$ the set of outgoing links of node u that are forwarding to the destination node. In line 14, we calculate the fraction f of demand (traversing and leaving the current node) by Γ. So, in line 19, we can calculate the proportion of load of each forward outgoing link, according to its value of Γ.

In the loop in lines 15-21, for each outgoing link of node u, the flow traversing the link is calculated according to its proportion of Γ. In line 24, the total load of each arc is updated with the partial loads calculated for destinations nodes $t \in T$. Finally, in line 26, the fitness value of the solution is computed.

The local search approach was also adapted. The dynamic shortest path graphs algorithm was used again, while the routing was recomputed from scratch each time there is a weight change in the graph. Time savings can be achieved with a dynamic version of the routing computation. Since under DEFT the traffic is split

Table 1: Instances used in computational experiments. The two-level hierarchical are based on models of [2, 9]. The random graphs have been generated using a fixed arc probability.

Instance	Nodes	Links	Capacities
2-level hierarchy	50	148	200 and 1000
2-level hierarchy	50	212	200 and 1000
Random topology	50	228	all 1000
Random topology	50	245	all 1000

among all forwarding links, a larger part of the graph is affected in comparison with OSPF. Thus, we believe the gains in terms of time are smaller when using dynamic routing in DEFT than in OSPF. The local search technique for OSPF described above has not been modified, i.e. the algorithm tries to reduce link congestion increasing weights in integer steps. Observe that this is not tuned for DEFT, which allows real weights, so the local search can miss an optimum increase. In particular, integer differences in path lengths rapidly lead to a negligible flow on that link. To counter this, we scale the path lengths before computing the flows according to the algorithm above.

5 Results

We have studied the performance our local search memetic algorithm on four different synthetic networks, each with seven different total demands. These are the same instances used in [5, 7] to keep our results comparable. Table 1 summarizes their characteristics.

We have run our memetic algorithm once with each instance and total demand, using the following parameters:

- Population size 100.
- Weights interval $[1, 20]$.
- 30 minutes of execution time.
- Probability of inheriting gene from elite parent during crossover: 0.7.
- Path scale factor 0.55.

All experiments have been conducted on a Pentium 4 2.54 GHz with 512 MB RAM running Linux. The results are shown in Figures 3 to 6. Each figure gives two metrics of the solution quality, the optimality gap in percent compared to the solution of OPT, described

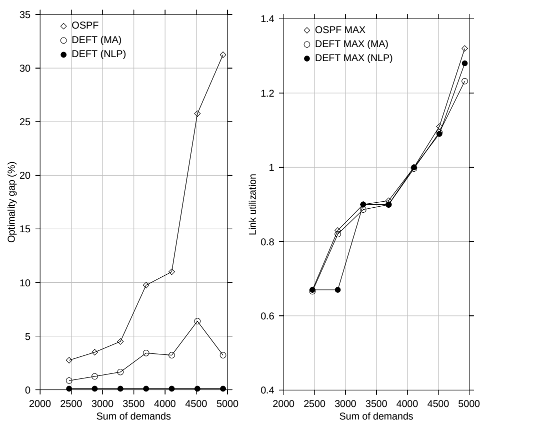

Figure 3: Comparison of optimality gap and maximum link utilization for the two-level hierarchy with 50 nodes and 148 links.

Table 2: Average running time per iteration of the non-linear solver and the MA for DEFT.

Instance	Nodes	Links	Time per iteration [s]	
			NLP	MA
2-level hierarchy	50	148	0.7–3.5	1.4–2.4
2-level hierarchy	50	212	1.0–4.8	2.3–2.8
Random topology	50	228	3.3–5.0	2.1–2.9
Random topology	50	245	6.0–12.3	2.7–3.8

in Section 1 and the maximum link utilization. The figures compare the results of the OSPF local search and the DEFT NLP solver, according to [7] and the results of our MA. Tables 2 and 3 show the average execution times per iteration and total number of iterations of the DEFT NLP solver and the MA.

The figures show the near optimality of weights found by the non-linear approach proposed in [7]. The MA improves substantially over OSPF and obtains good results in less time. In practice, the figures show the advantage of splitting the flow among shortest and non-shortest paths from origin to destination nodes. In Figure 3 we see the high cost OSPF results for a two-level hierarchical network. In the same network the MA finds near-optimal results within 7% of those found by the NLP DEFT approach, and far better than OSPF. Looking at the other instances we observe similarly that the MA using DEFT can improve over the results of OSPF, but not reaches the quality of the DEFT NLP solver.

Considering the maximum link utilization, all algorithms show comparable results. In general, DEFT can improve over OSPF, with the maximum utilization of

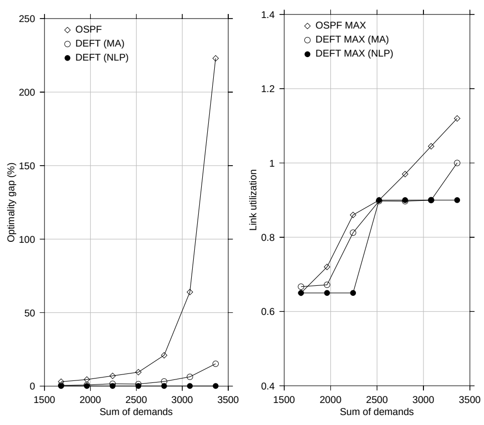

Figure 4: Comparison of optimality gap and maximum link utilization for the two-level hierarchy with 50 nodes and 212 links.

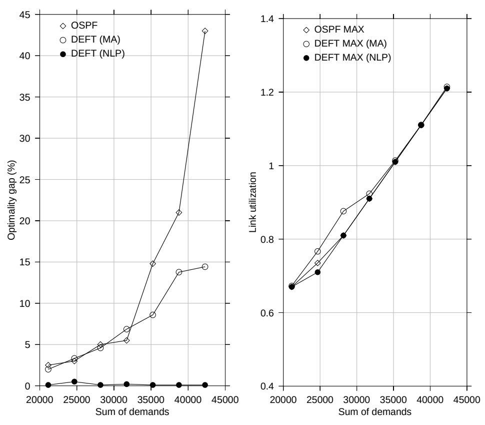

Figure 5: Comparison of optimality gap and maximum link utilization for a random topology with 50 nodes and 228 links.

Table 3: Total number of iterations of the non-linear solver and the MA for DEFT.

Instance	Nodes	Links	# iterations	
			NLP	MA
2-level hierarchy	50	148	271–825	760–1279
2-level hierarchy	50	212	308–1020	639–800
Random topology	50	228	400–1400	631–861
Random topology	50	245	620–1400	483–672

the DEFT NLP solver slightly better than that of the MA. All graphs of the maximum utilization show a knee when the demand approaches 90% link utilization. Given the properties of the objective function depicted in Figure 1, there results a huge increment in the cost when the demand surpasses a certain threshold. In practice, that threshold in congestion varies according the topology, the demand and the weights set, but is clearly observable in the objective function as the utilization approaches the manageable congestion. That objective function's main property is to describe the growing probability of congestion by the increase in traffic. With more demand leading to a link utilization above the 90% threshold, we see the expected near optimality of DEFT followed by MA with a large gap to OSPF.

6 Conclusions and future work

Based on the preliminary results presented in the previous section, we can conclude that the MA obtains better results under DEFT than under OSPF. The MA using DEFT can always improve over OSPF, in some cases up to a factor of six. However, the two-stage approach proposed in [7] obtains better results when compared with the adapted MA proposed in this paper, spending comparable running times.

Observe that the results are preliminary, from a method adapted from OSPF to DEFT. We think that there is a lot of potential for improving the quality, as well as the running times of the current MA. Specifically, the algorithm is not dynamic, and makes no use of real weights. Running time could be improved substantially if using dynamic procedures for flow computation.

Recently, [8] have proposed a path-based protocol improving over DEFT. We also intend to investigate this new approach.

Another interesting line of research where local search based methods can be useful is survivable network design, where optimization solvers cannot be applied directly.

References

[1] L. Buriol, M. Resende, C. Ribeiro, and M. Thorup. A hybrid genetic algorithm for the weight setting problem in OSPF/IS-IS routing. *Networks*, 46(1):36–56, 2005.

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

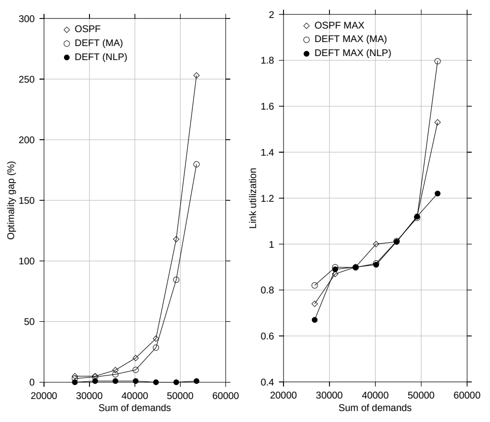

Figure 6: Comparison of optimality gap and maximum link utilization for a random topology with 50 nodes and 245 links.

[3] M. Ericsson, M. Resende, and P. Pardalos. A genetic algorithm for the weight setting problem in OSPF routing. *J. of Combinatorial Optimization*, 6:299–333, 2002.

[4] B. Fortz and M. Thorup. Internet traffic engineering by optimizing OSPF weights. In *INFOCOM 2000*, pages 519–528, 2000.

[5] B. Fortz and M. Thorup. Increasing internet capacity using local search. *Computational Optimization and Applications*, 29(1):13–48, 2004.

[6] John T. Moy. *OSPF: Anatomy of an Internet Routing Protocol*. Addison-Wesley, 1998.

[7] Dahai Xu, Mung Chiang, and Jennifer Rexford. DEFT: Distributed exponentially-weighted flow splitting. In *INFOCOM 2007*, pages 71–79, May 2007.

[8] Dahai Xu, Mung Chiang, and Jennifer Rexford. Link-state routing with hop-by-hop forwarding can achieve optimal traffic engineering. In *Submitted to INFOCOM 2008*, 2008.

[9] 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.