

# A Memetic Algorithm for OSPF and DEFT routing to minimize network congestion

Roger Reis<sup>1</sup>, Marcus Ritt<sup>1</sup>, Luciana S. Buriol<sup>1</sup>, Mauricio G. C. Resende<sup>2</sup>

<sup>1</sup>*Universidade Federal do Rio Grande do Sul - Instituto de Informática  
Av. Bento Gonçalves, 9500 - Campus do Vale - Bloco IV - Bairro Agronomia  
Porto Alegre - RS - Brazil  
{rsreis,mrpritt,buriol}@inf.ufrgs.br*

<sup>2</sup>*AT&T Labs Research  
180 Park Avenue, Room C241, P.O. Box 971  
Florham Park, NJ, USA  
mgr@research.att.com*

## Abstract

---

Interior gateway routing protocols like OSPF (*Open Shortest Path First*) and DEFT (*Distributed Exponentially-Weighted Flow Splitting*) send flow through forward links towards the destination node. OSPF routes only on shortest-weight paths, whereas DEFT sends flow on all forward links, but with an exponential penalty on longer paths. Finding suitable weights for these protocols is known as the *weight setting problem*. In this article, we present a memetic algorithm for the weight setting problem using both protocols. The algorithm uses dynamic flow and dynamic shortest path computations. We report computational experiments that show that DEFT achieves less network congestion when compared with OSPF, while presenting a larger delay.

*KEYWORDS:* Telecommunication networks, routing, metaheuristics.

---

## 1 Introduction

The Internet is composed of numerous Autonomous Systems (ASes), each one using an Interior Gateway Protocol (IGP) to control routing within the AS. The topology of an Internet network is given by a directed graph comprised of a set of routers (nodes) and a set of communication links (arcs). A set of routers under the control of one or more network operators who apply the same routing policy is what characterizes an AS. Given a demand matrix containing the amount of traffic to be sent between all pairs of routers, IGP routing protocols establish rules on how the load will be sent from source to destination in the AS.

Certainly, the most flexible routing model is the fractional multi-commodity flow routing (in Section 2 we refer to this routing as OPT). In this routing model, the load of each demand pair is routed independently of the other demand pairs, making the best use of the network link capacities. However, it is difficult to implement this protocol in practice because of arbitrarily long paths, and the arbitrarily small demand loads that can be routed on high-capacity links. Therefore, telecommunication network protocols are based on routing models that are less efficient with respect to capacity utilization, but easier to implement in practice.

Among existing IGP routing protocols, there are a few that have been long used in practice, as is the case of OSPF (*Open Shortest Path First*), whereas others have been recently proposed and are not yet implemented in real networks, as is the case of DEFT (*Distributed Exponentially-Weighted Flow Splitting*). Both OSPF and DEFT are link-state routing protocols. These protocols allow the network operator to calculate paths on the network by setting adequate link weights to balance the load on the network. The load is then sent through the paths from source to destination and quantities such as network congestion, link utilization, and delay can be measured. The problem of finding proper weights to optimize an objective function on these metrics is known as the *Weight Setting Problem* (WSP).

The objective of intradomain traffic engineering is the efficient utilization of the available network resources within an AS under traffic constraints. With this aim, the weight setting problem has been studied for almost

a decade for OSPF routing. In OSPF, integer link weights are set by the network operator. The flow is routed through the shortest paths, splitting traffic evenly, in each node, among all outgoing shortest path links.

The weight setting problem in OSPF routing was proved to be NP-hard by Fortz and Thorup [8]. They also proposed a tabu search heuristic, and compared results with a lower bound for the problem obtained by solving the corresponding multi-commodity flow problem. Their results show that for most instances, the solutions are about 10% above the lower bound. But for a few instances, the gaps between the solution and the lower bound were up to twice the lower bound.

Another heuristic, a genetic algorithm (GA) introduced by Ericsson et al. [6], was applied on the same set of instances considered in Fortz and Thorup [8], and also produced large gaps for the hardest instances. Buriol et al. [5] incorporated a local search procedure into a modified version of the GA of Ericsson et al., resulting in a memetic algorithm. For most of the same instances, the gap decreased when compared with results from the previous methods. A mathematical formulation for the problem was first proposed by Holmberg and Yuan [10]. Subsequently, survivability aspects were introduced by Broström and Holmberg [4].

A wide range of other link-state routing protocols exist. One of these protocols is IS-IS (Intermediate System to Intermediate System). It considers basically the same rules as OSPF, with the main difference that OSPF is an IP protocol, whereas IS-IS is natively an ISO (International Organization for Standardization) network layer protocol. A few different link-state protocols are reported in the literature. Examples of these protocols are those that consider unsplittable shortest path routing [2, 3].

DEFT, or Distributed Exponentially-Weighted Flow Splitting, is an IGP routing protocol, recently proposed by Xu et al. [15]. DEFT considers not only shortest paths for routing. It directs flow through all forward paths, but with exponential costs for longer paths. Furthermore, DEFT considers a real-weight representation. The protocol was designed to simplify intradomain traffic engineering. Subsequently, Xu et al. [16] proposed PEFT (*Penalizing Exponential Flow-splitting*), a path-based routing protocol that splits traffic over multiple paths with an exponential penalty on longer paths. The main difference between these two protocols is that, in terms of flow splitting, DEFT is a link-based protocol, whereas PEFT is a path-based protocol. In PEFT, the outgoing flow at any node is split among all shortest paths to a destination node, whereas in DEFT, the outgoing flow is split among all shortest path links. If an outgoing shortest path link belongs to more than one shortest path, in PEFT this link receives more flow than a shortest path link that belongs to only one shortest path to the destination node.

For an overview of traffic engineering, the survey by Wang et al. [14] presents more than one hundred up-to-date references on different aspects of optimization in Internet routing.

This paper compares the memetic algorithm presented in [5] on DEFT and OSPF, two link-state routing protocols. The algorithm, originally proposed for OSPF, is adapted for DEFT. The dynamic shortest path and dynamic flow distribution for DEFT are presented in detail. The most common dynamic shortest path algorithms used in practice were proposed by Ramalingam and Reps [13], whereas to the best of our knowledge, dynamic OSPF routing update was first proposed by Fortz and Thorup [8]. In this paper, we consider an implementation of DEFT with integer weights, a constrained variant of the real-weight representation originally proposed by Xu et al. [15]. One reason for this is that routers currently used in practice only allow integer weights and therefore applying DEFT with integer weights could be a first step in the direction of using DEFT in practice. In the remainder of this paper, we refer to DEFT with integer weights as *int-DEFT*.

The paper is organized as follows. Section 2 presents a description of the general routing problem. In Section 3, the OSPF and DEFT protocols are described in detail. Section 4 presents the proposed memetic algorithm for the OSPF WSP. Section 5 introduces modifications needed to deal with DEFT routing. Computational results are presented in Section 6 and concluding remarks are made in Section 7.

## 2 The general routing problem

Let  $G = (V, E)$  be a directed graph modeling a network with routers  $V$  and links  $E \subseteq V \times V$ . Each link  $(u, v) \in E$  has a flow capacity  $c_{u,v}$ . Let  $D$  be a demand matrix, where  $D_{u,v}$  denotes the traffic flow from source node  $u$  to target node  $v$  for  $u, v \in V$ . Let  $T \subseteq V$  be the subset of all target nodes in  $V$ , i.e.,  $T = \{v | D_{uv} > 0\}$ . The general routing problem can be stated as to find the flows  $f_{u,v}$  on each arc  $(u, v) \in E$  that minimize an objective function and delivers all demands.

Thus, we have

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

where  $\Phi$  is the network-link cost function, which depends on the current flow and the link capacity. Typically,  $\Phi$  is the piecewise linear function

$$\Phi(f_{u,v}, c_{u,v}) = \begin{cases} f_{u,v} & f_{u,v}/c_{u,v} < 1/3 \\ 3f_{u,v} - 2/3c_{u,v} & 1/3 \leq f_{u,v}/c_{u,v} < 2/3 \\ 10f_{u,v} - 16/3c_{u,v} & 2/3 \leq f_{u,v}/c_{u,v} < 9/10 \\ 70f_{u,v} - 178/3c_{u,v} & 9/10 \leq f_{u,v}/c_{u,v} < 1 \\ 500f_{u,v} - 1468/3c_{u,v} & 1 \leq f_{u,v}/c_{u,v} < 11/10 \\ 5000f_{u,v} - 16318/3c_{u,v} & 11/10 \leq f_{u,v}/c_{u,v} \end{cases} \quad (2)$$

proposed in [8, 9] and shown in Figure 1.

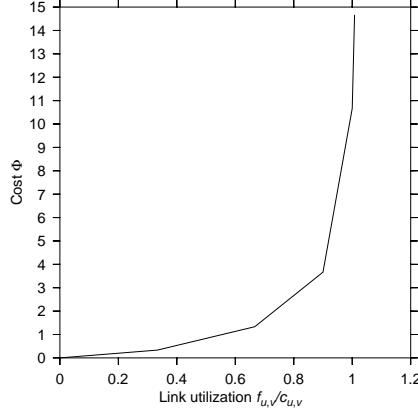


Figure 1: Link cost  $\Phi$  as a function of the link utilization  $f_{u,v}/c_{u,v}$ .

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

$$\sum_{w:(v,w) \in E} f_{v,w}^t - \sum_{u:(u,v) \in E} f_{u,v}^t = D_{v,t} \quad (3)$$

and the individual flow aggregation

$$f_{u,v} = \sum_{t \in T} f_{u,v}^t. \quad (4)$$

As the constraints and the objective function are linear, the optimum solution can be obtained by solving the linear program OPT defined by equations (1), (3), (4), and the trivial constraints  $0 \leq f_{u,v}^t, 0 \leq f_{u,v}$ . The optimal solution of OPT is a lower bound for the cost of any routing protocol.

### 3 OSPF and DEFT routing protocols

The OSPF protocol uses weights  $w_{u,v}$  on links  $(u, v) \in E$  to determine the flow distribution of demands. The weights are 16-bit integers in the range  $[0, 2^{16} - 1]$ . Each router maintains a link-state database of the network topology and the weights, and regularly exchanges state information with other routers in the same AS to keep the database current. To route incoming traffic, a router maintains a shortest path graph using the weights as distances to all known target nodes within the AS. The outgoing traffic of a node  $u$  with destination  $t$  is split equally among all outgoing links on shortest paths to  $t$ .

The DEFT protocol relaxes this shortest-path-only constraint and also allows routing on non-shortest paths. The outgoing traffic of a node  $u$  is split proportionally among all forward links to a target node  $t$ . Links belonging to non-shortest paths receive exponentially greater penalties, and consequently receive smaller flow amounts to carry. Formally, let  $d_u^t$  be the distance from node  $u$  to destination node  $t$ . Then  $h_{u,v}^t = d_v^t + w_{u,v} - d_u^t$  is the difference between the length of the shortest path and the length of the path traversing link  $(u, v)$ . The non-normalized flow fraction  $\Gamma$  in the direction to target node  $t$  passing by link  $(u, v)$  is defined as

$$\Gamma(h_{u,v}^t) = \begin{cases} e^{-h_{u,v}^t} & \text{if } d_u^t > d_v^t \\ 0 & \text{otherwise} \end{cases} \quad (5)$$

and the fraction of the total flow  $\Gamma(h_{u,v}^t) / \sum_{v:(u,v) \in E} \Gamma(h_{u,v}^t)$  is calculated for each outgoing link  $(u, v)$  of  $u$ . According to [15], in terms of total link cost and maximum utilization, there always exists a weight setting such that DEFT has a smaller total cost than OSPF.

To observe the difference that the link weight setting has in both protocols, consider the example in Figure 2. It shows a network with a link weight increase in arc  $(b, t)$  from weight  $P_2$  (left) to weight  $P_2'$  (right). Consider that  $P_1 > P_2' > P_2 > 0$  and arcs  $(u, a)$  and  $(u, b)$  have the same positive weight. In OSPF, the traffic from node  $u$  to  $t$  is routed through the shortest path  $u-b-t$ . Node  $a$  does not receive any flow. The weight change does not alter this scenario.

However, when routing with DEFT, node  $a$  receives a certain flow fraction and the change in the weight of arc  $(b, t)$  represents a modification of this fraction. Increasing the weight of  $(b, t)$  causes a decrease in the amount of flow routed through  $b$ , and a larger part of the flow is now routed through  $a$ .

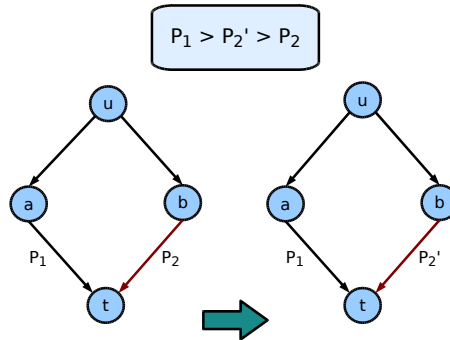


Figure 2: Effect in OSPF and DEFT routing when increasing the weight of arc  $b-t$ .

The WSP for OSPF was formulated in [8] and different heuristics [6, 5] have been explored for its solution. DEFT was proposed in [15], and to the best of our knowledge, no heuristic other than the memetic algorithm described in this paper has been proposed to solve the WSP for DEFT. The authors of the DEFT protocol proposed a two-stage iterative algorithm, based on non-linear and non-smooth optimization, for DEFT with real-valued weights.

## 4 A memetic algorithm for OSPF

A memetic algorithm (MA), 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. Memetic algorithms were introduced in [12]. In the context of a MA, 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*. An MA is a populational method in which, during each iteration, individuals are combined through a crossover procedure to generate 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 the case of the WSP, is the network congestion function (1).

In our algorithm, each individual is represented by a vector of integer arc weights. The initial population consists of individuals with randomly generated weights. The population is structured into three classes, according to their fitness, as introduced by Bean [1] for sequencing problems, and later applied by Ericsson et al. [6] in a genetic algorithm for OSPF routing. Class  $\mathcal{A}$  is composed of the 25% fittest individuals, class  $\mathcal{C}$  is composed of the 5% less fit solutions, and the remaining population composes class  $\mathcal{B}$ . The solutions from class  $\mathcal{A}$  pass directly (i.e. without a change) to the next generation. The class  $\mathcal{C}$  solutions are generated at random using the same generation process used for the initial population. The remaining solutions are replaced by solutions generated by a 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 [11] that prioritizes (with 70% probability) genes from parents in class  $\mathcal{A}$ . With a small probability of 1%, the child inherits a completely random allele at some given gene. A local search heuristic is applied to each solution generated by crossover. The local improvement procedure examines the effect of increasing the weights of a subset of the arcs. The candidate arcs are all arcs whose weight is smaller than the maximum weight allowed, and the candidates are visited in decreasing order of their routing cost  $\Phi(f_{u,v}, c_{u,v})$ . To reduce the routing cost of a candidate arc, the procedure attempts to increase its weight to reduce its load. If this leads to a reduction of the overall routing cost, the change is accepted, and the procedure is restarted. Otherwise, the increase is rejected and the procedure continues with the next candidate arc.

The local search is repeated until  $k$  candidate arcs have their weight increased without improving the solution. We tested different values of  $k$  and decided to set  $k = 5$ , as in the original procedure proposed in [5] to explore a small neighborhood. Keeping the neighborhood small helps to preserve the diversity of the population.

The local search performs several expensive solution evaluations. 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 next section describes the adaptation of this algorithm for dealing with DEFT routing.

## 5 A memetic algorithm for DEFT routing

In this section we present the above described memetic algorithm modified for DEFT. We maintained the entire memetic algorithm structure as well as the operators described in the previous section, only changing the routing procedure.

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

Opposed to OSPF, the weights in DEFT are positive real numbers. Therefore, an implementation of DEFT on current routing hardware, has to decide how to map the real weights onto the available range of weights, typically a 16-bit integer. Another issue is how to handle small flow fractions. Even a path considerably longer than the shortest path to the target, receives a flow, but this flow can be tiny, since the assigned fraction of flow decreases exponentially. Distributing flow to much longer paths can increase communication latency.

To obtain a realistic implementation, we solve these problems following the proposal given in [15, section II.C]. Our implementation works with integer weights, but uses a scaling parameter  $p$ . Actual read-valued distances are obtained by dividing the integer distances by  $p$ . We call our implementation int-DEFT where necessary to avoid ambiguity. In the experiments described later in this paper, we use a scaling parameter of  $p = 1.8$ .

To avoid routing on long paths with a marginal flow contribution, we introduce a maximum gap  $g$ , and route flow only on links whose integer gap  $h_{u,v}^t$  is at most  $g$ . In the experiments below we use  $g = 9$ . This excludes from routing paths which would receive a fraction of the flow having less than  $e^{-10/1.8} \approx 0.39\%$  of the flow routed on a shortest path. Observe that for the special case of a maximum gap equal to zero, DEFT routing reduces to OSPF routing.

In summary, equation (5) is modified to

$$\Gamma(h_{u,v}^t) = \begin{cases} e^{-h_{u,v}^t/p} & \text{if } d_u^t > d_v^t \text{ and } h_{u,v}^t \leq g \\ 0 & \text{otherwise} \end{cases} \quad (6)$$

Figure 3 describes in pseudocode how DEFT rules are implemented. As in OSPF, for each destination node  $t \in T$  we compute the reverse shortest path distance (line 2) and, with a scan of the arcs, the shortest path graph  $G^t$  (line 3). In lines 4–22 we detail the procedure `ComputePartialLoads` that implements the DEFT rules that allows flow be routed on non-shortest paths. In line 5 we sort the nodes in decreasing order of their distances to  $t$ . Nodes are analyzed one by one, in decreasing distance to the target node. The loop in lines 8–13 calculates the sum ( $\Gamma_{total}$ ) of the flow distribution function (6) 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  $u$ . In line 14, we calculate the total demand  $f$  (traversing and leaving the current node) per unit of  $\Gamma$ . In the loop in lines 15–21, for each forward outgoing link of node  $u$ , the flow traversing the link is calculated according to its proportion of  $\Gamma$ . In line 24, the total load of each arc is updated with the partial loads calculated for destination 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 graph algorithm, as well as dynamic flow computation were adapted for DEFT. The general idea of the local search technique for OSPF described in the previous section has not been modified, i.e. the algorithm tries to reduce link congestion increasing weights in integer steps.

Observe that the local search is not tuned for DEFT, since it works with integer weights and therefore only considers weight changes with granularity  $1/p$ , where  $p$  is the scale parameter described above. Therefore, the local search can miss an optimum increase.

### 5.1 The dynamic flow algorithm for DEFT

The DEFT routing protocol distributes the flow among shortest and non-shortest paths to a target node. Small changes can cause a new flow distribution in the network, even when the shortest path graph is unaltered. As

```

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

```

Figure 3: Pseudocode describing solution evaluation for DEFT.

a consequence, we can expect that a change in an arc weight will lead to an altered flow in a larger number of links. To minimize the computational cost of evaluations of the objective function by the local search procedure, we developed a *dynamic flow update algorithm* for int-DEFT. This algorithm receives a unitary increment of a single arc weight and updates only the part of the network affected by this change. There are three main cases to be analyzed. Let  $t$  be the target node. Given a unitary increment  $\Delta$  in the weight of arc  $(u, v)$ , nodes can be classified in three different cases according to their distance change. First, nodes with no outgoing load (leaving or traversing the node) directed to  $t$  are not affected by the increment  $\Delta$ . In the second case, nodes whose distances have changed and that reach  $t$ , before and after the change, only through arc  $(u, v)$  have no modifications in their loads. The loads traversing these nodes are affected equally by any  $\Delta$  variation of the arc weight. Therefore, the flow distribution is unaltered. In the third case are those nodes that, with increment  $\Delta$ , create alternative paths that do not traverse arc  $(u, v)$ . Those, and every intermediate successor node, have their flow distribution altered and, therefore, have to be reevaluated. Consider, as an example, Figure 4. Suppose that due to the increment  $\Delta$  in arc  $(u, v)$ , nodes  $u$  and  $a$ , that are above the dashed line, are the nodes whose distances are affected. Suppose that, before the increment, arc  $(a, b)$  did not belong to the shortest path, but after the increment it does. In this case, some load is sent through arcs  $(a, b)$  and then  $(b, v)$ , while links  $(a, u)$  and  $(u, v)$  have their loads decreased.

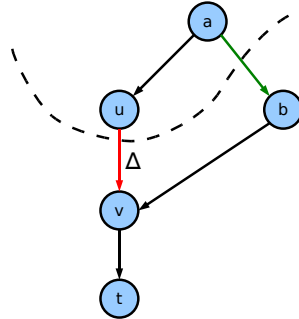


Figure 4: Dynamic flow update.

Figure 5 presents the pseudocode for solution evaluation using the dynamic flow updating procedure. In line 1, the current distance vector is preserved. In lines 3–4, for each target node  $t \in T$  the reverse shortest path graph  $G^t$  is calculated. The loop in lines 6–13 identifies all nodes  $u$  with two or more forward outgoing arcs having at least one successor whose shortest distance to  $t$  is not altered and a different successor whose shortest distance to  $t$  is increased by exactly  $\Delta$ . Those nodes represent the third case described above, and their flows

will be altered. They are stored in a heap data structure  $H_{d,d_{old}}$ , whose keys are pairs of the current and the previous shortest distance (before the  $\Delta$  increase) to  $t$ . The heap is ordered by the current distance and, in case of a tie, by the previous distance. More formally,

$$(d, d_{old}) < (d', d'_{old}) \quad \text{iff} \quad d < d' \text{ or } (d = d' \text{ and } d_{old} < d'_{old}) \quad (7)$$

```

procedure FlowDistributionDEFT( $G = (V, E), T, D, (w_e)_{e \in E}, (d_e)_{e \in E}, \Delta$ )
1   $d_{old} = d$ 
2  for  $\forall t \in T$ 
3     $d = \text{ReverseDijkstra}(G, t, w)$ 
4     $G^t = \text{ComputeShortestPathGraph}(G, w, d)$ 
5     $H_{d, d_{old}} = \emptyset$  // order according to equation (7)
6    for each  $u \in V$ 
7      condition1 = condition2 = false
8      for each  $v \in \text{OUT}(u)$ 
9        if  $d_v^t = d_{old, v}^t + \Delta$  then condition1 = true
10       if  $d_v^t = d_{old, v}^t$  then condition2 = true
11     endfor
12     if condition1 and condition2 then InsertHeap( $H_{d, d_{old}}, u$ )
13   endfor
14   for each  $p \in H_{d, d_{old}}$ 
15     for each  $q \in \text{OUT}(p)$ 
16       if  $q \notin H_{d, d_{old}}$  then InsertHeap( $H_{d, d_{old}}, q$ )
17     endfor
18   endfor
19   for each  $u \in H_{d, d_{old}}$ 
20      $\Gamma_{total} = 0$ 
21     for each  $v \in \text{OUT}(u)$ 
22       if  $d_u^t > d_v^t$  and  $h_{u, v}^t \leq g$  then
23          $h_{u, v}^t = d_v^t + w_{u, v} - d_u^t$ 
24          $\Gamma_{total} = \Gamma_{total} + e^{-h_{u, v}^t}$ 
25       endif
26     endfor
27      $f = D_{u, t} + \sum_{v: (v, u) \in G^t} f_{(v, u)}^t / \Gamma_{total}$ 
28     for each  $v \in \text{OUT}(u)$ 
29       if  $d_u^t > d_v^t$  and  $h_{u, v}^t \leq g$  then
30          $\gamma = e^{-h_{u, v}^t}$ 
31          $f_{u, v}^t = f \gamma$ 
32       else
33          $f_{u, v}^t = 0$ 
34       endif
35     endfor
36   endfor
37   for each  $(u, v) \in E$ 
38      $f_{u, v} = f_{u, v} + f_{u, v}^t$ 
39 endfor
40  $\Phi = \sum_{(u, v) \in E} \phi(u, v)$ 
end FlowDistributionDEFT.

```

Figure 5: Pseudocode of the dynamic flow algorithm.

The loop in lines 14–18 adds every intermediate node which is a successor in direction of  $t$  of a node whose flow altered to the heap. The new flow distribution is calculated in lines 19–37. Line 27 calculates the total flow leaving the current node. In lines 28–36, this flow is split proportionally among all outgoing arcs. In line 38, the sum of all flow fractions is calculated and, finally, in line 40, the total network congestion  $\Phi$  is evaluated.

Since under DEFT the traffic is split among all forwarding links, a larger part of the graph is affected as compared with OSPF. Thus, our hypothesis that the time savings are smaller when using dynamic routing in DEFT than in OSPF were confirmed in the computational experiments.

## 6 Experimental results

We have studied the performance of our memetic algorithm on twelve synthetic networks and another instance with real data from a large tier-1 Internet service provider (ISP). These are the same instances used in [9, 15] to keep our results comparable. Table 1 summarizes their characteristics. The columns represent, respectively, the instance class, the instance name, number of nodes, number of arcs, the values of link capacities (instance `att` has a large amount of different values, and so it was omitted from the table), and the number of origin-destination (o-d) demand pairs. The instances are classified in four classes: historical data from the ISP Backbone (`att`), 2-level hierarchical networks (`hier`), random networks (`rand`), and Waxman networks (`wax`). Further details about the instances can be found in [5, 8].

Table 1: Instances used in computational experiments.

Instance	Name	Nodes	Links	Capacities	Pairs o-d
ISP backbone	att	90	274	-	272
2-level hierarchy	hier50a	50	148	200 and 1000	2450
	hier50b	50	212	200 and 1000	2450
	hier100	100	279	200 and 1000	9900
	hier100a	100	360	200 and 1000	9900
Random topology	rand50	50	228	1000	2450
	rand50a	50	245	1000	2450
	rand100	100	403	1000	9900
	rand100b	100	503	1000	9900
Waxman	wax50	50	169	1000	2450
	wax50a	50	230	1000	2450
	wax100	100	391	1000	9900
	wax100a	100	476	1000	9900

We used seven different demand matrices for each network, obtained by scaling a basic demand matrix for each instance by a factor from 6 to 12. The memetic algorithm has been run with the following parameters

- Population size: 50 individuals,
- Weight interval:  $[1, 20]$ ,
- 60 minutes of execution time,
- Probability of inheriting gene from elite parent during crossover: 0.7,
- Maximum gap  $g$  for  $h_{u,v}^t$ : 9,
- Scale parameter  $p$ : 1.8.

We tested different values for these parameters, without relevant changes in the results. Thus, we decided to preserve the same parameters used in the original algorithm for OSPF [5].

All experiments were made on a cluster of 10 Intel Duo Core processors with 1.23 GHz, 1.0 GB RAM, and running Linux 2.6.18-4. Each run used a single processor.

We performed several analyses of the results. In the following section, we study the time savings obtained by the dynamic flow algorithm for DEFT. In the subsequent sections, we evaluate the solution quality in terms of network congestion and three additional metrics. For each experiment, we report the average of five runs of our memetic algorithm for each combination of network and total demand.

## 6.1 Experimental evaluation of the dynamic flow algorithm

This experiment explores the time savings obtained by tuning the implementation to work efficiently for DEFT and by using the dynamic flow algorithm described in subsection 5.1 compared to a straightforward adaptation of the original algorithm proposed for OSPF. We compare three different implementations:

**int-DEFT** A straightforward adaptation to DEFT of the memetic algorithm proposed for OSPF. It uses a dynamic single source shortest path (DSSSP) algorithm. The routing follows DEFT rules and is computed statically.

**int-DEFT-opt** int-DEFT, with its data structures tuned to work efficiently with DEFT.

**int-DEFT-DF** int-DEFT-opt, but additionally using the dynamic flow algorithm.

Figure 6 shows a comparison of the execution time in hours for 1000 generations of the three implementations and Table 2 shows the speedups of the improved implementations over int-DEFT. All 13 instances were tested with the highest total demand.

The straightforward adaptation from OSPF is in average 3.12 times slower than the final version. From the data in Table 2 we can also see, that the performance gains are mainly because of the dynamic flow computation. Tuning the implementation for DEFT results only in an average speedup of 1.27, while the dynamic flow computation algorithm was responsible for 51% to 68% of the time savings, for the set of instances tested. Thus, we conclude that even for non-shortest path routing protocols like it is worthwhile to implement dynamic flow computation.

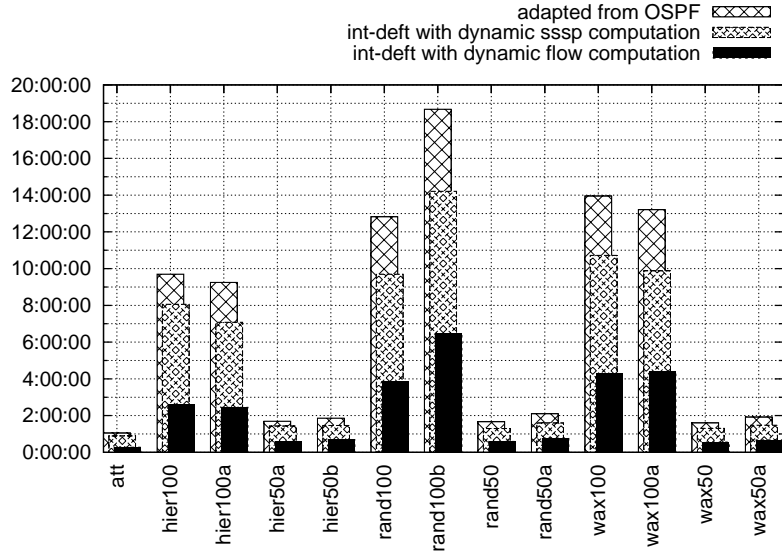


Figure 6: Execution time of 1000 generations for three DEFT routing implementations.

Table 2: Speedup of int-DEFT-opt and int-DEFT-DF over int-DEFT.

Implementation	Instance							
	att	hier100	hier100a	hier50a	hier50b	rand100	rand100b	
int-DEFT-opt	1.18	1.20	1.31	1.19	1.27	1.32	1.31	
int-DEFT	3.74	3.73	3.77	2.89	2.71	3.33	2.89	
Implementation	rand50	rand50a	wax100	wax100a	wax50	wax50a		
	int-DEFT-opt	1.29	1.32	1.30	1.34	1.22	1.30	
int-DEFT	2.81	2.68	3.25	3.01	2.86	2.82		

## 6.2 Quality of solutions

The second set of experiments compares the quality of the solutions obtained by the memetic algorithm when routing with OSPF and DEFT. We report the optimality gap, i.e. the additional routing cost of the best solution found, as a percentage of the routing cost of the lower bound given by the solution of OPT. Figures 7 to 11 show the results for DEFT and OSPF and all tested instances.

For the six instances in Figures 7 and 8, DEFT is able to improve over OSPF. In particular for high total demands, where OSPF has large optimality gaps, DEFT can lower the gaps considerably.

For the seven instances, shown in Figures 9 to 11, both DEFT and OSPF result in about the same optimality gap. For the last five instances in Figures 10 and 11, there are several cases, where DEFT even yields slightly worse results than OSPF.

## 6.3 Analysis of the routing paths

The third experiment we performed aims at analyzing the number of intermediate nodes involved in the routes of a demand path. For the demand matrix, we compare the following three metrics:

1. Path length: the average path length over all paths used for routing and all o-d demand pairs, measured in number of nodes of the path.
2. Number of paths: the average number of different paths used for routing the o-d demand pairs. Two paths are considered different if one path has at least one arc that does not belong to the other path.
3. Percentage of intermediate nodes affected: the average number of intermediate nodes routing a o-d demand pair, as a percentage of the total number of nodes.

For each instance, we present the minimum, maximum, and average values, as well as the standard deviation, considering all paths of all o-d demand pairs. The values are the average of three runs of 1000 generations each. To conduct this experiment, we selected four instances: att, hier100, hier50a, and wax100. For each instance, we considered four demand matrices. The next subsections explore each one of the metrics.

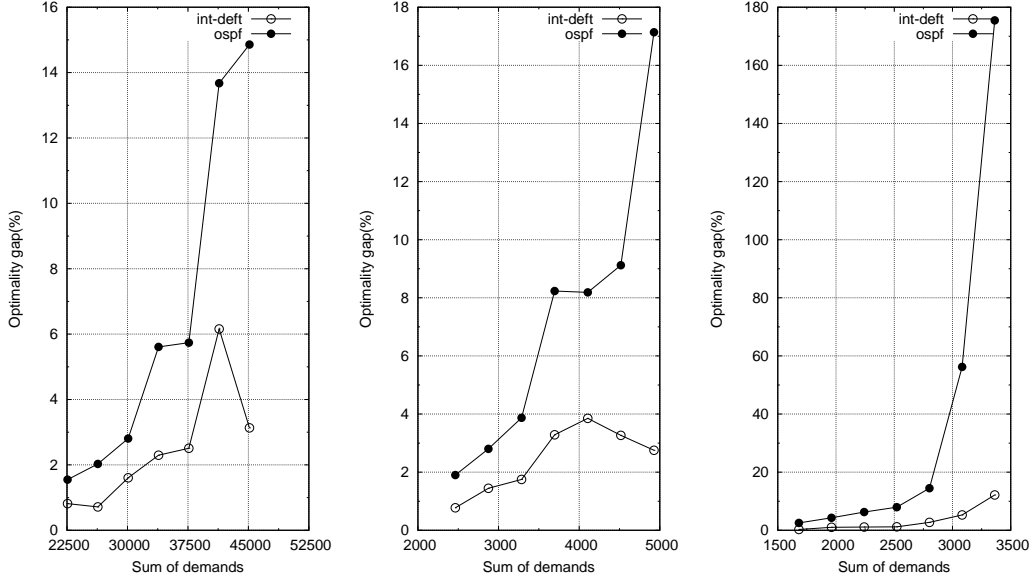


Figure 7: MA optimality gap for instances att, hier50a, and hier50b measured for OSPF and DEFT solutions.

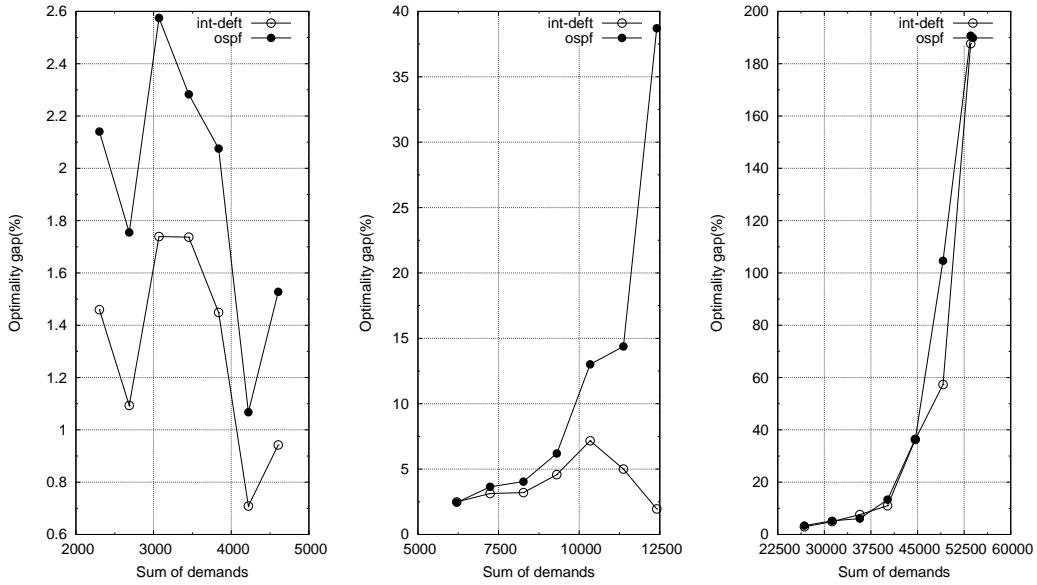


Figure 8: MA optimality gap for instances hier100, hier100a, and rand50a measured for OSPF and DEFT solutions.

### 6.3.1 Path length

This experiment has the objective of comparing the network delay for int-DEFT and OSPF measured as the length of the paths, i.e., the number of nodes that comprises the paths. We report the shortest path, the longest path, and the average path size among all paths of all o-d demand pairs. The measure is calculated for int-DEFT and OSPF for the best solution of the 100th generation, for four demand matrices of the four instances used in the experiments. Figure 12 presents the results for int-DEFT and OSPF.

From the plots, it is possible to observe that the path lengths in int-DEFT are about 40% longer than in OSPF. For example, for the demand matrix with the largest sum of demands, the average path lengths found by int-DEFT are 10.63, 12.80, 8.2, and 6.92 hops, whereas the corresponding values for OSPF are 7.94, 9.03, 6.01, and 4.47 hops. If we compare the path length with the shortest possible length as given by the topology of the instances, OSPF adds in average 2 hops, compared to 4.8 hops in int-DEFT. In a telecommunication network, it is desirable to maintain the path lengths as short as possible. One reason is that with the path length, the expected number of demands pairs affected by a failure increases. Thus, the length of a path is directly related to the quality of service of a telecommunication network.

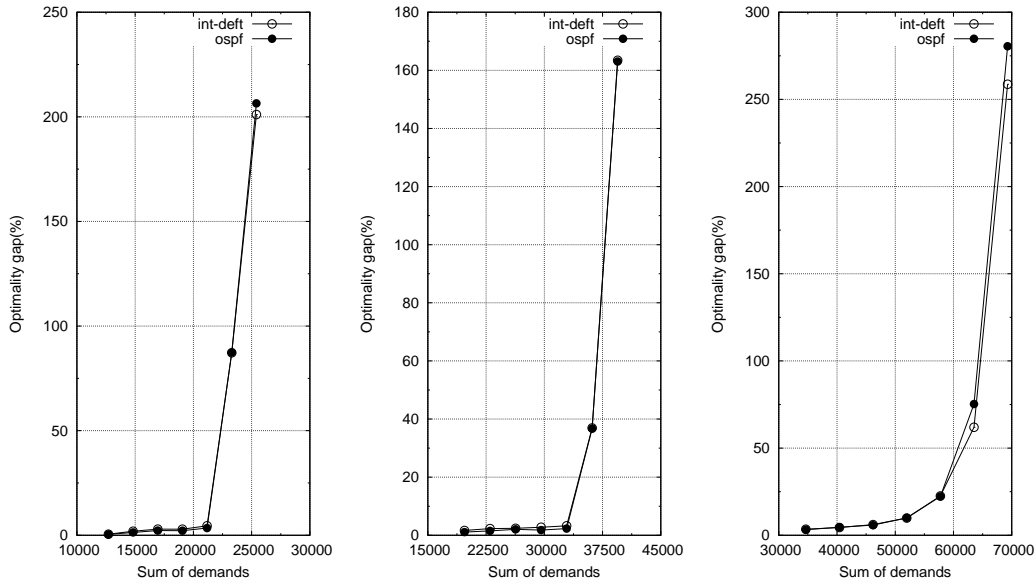


Figure 9: MA optimality gap for instances wax50, wax50a and rand100 measured for OSPF and DEFT solutions.

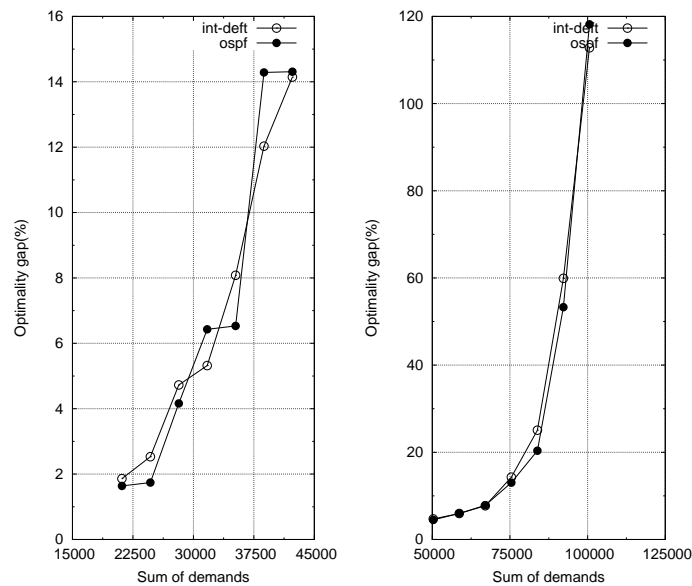


Figure 10: MA optimality gap for instances rand50 and rand100b measured for OSPF and DEFT solutions.

With respect to the minimum path length observed, they are the same for OSPF and int-DEFT. The minimum value is four for instance att, and two for the other three instances. That was expected, since both protocols route through the shortest path, and a path of length two indicates that the path is composed of a single direct link. Instance att has no o-d demand pair between all pairs of nodes, while the other instances have. Thus, it is possible to have the minimum value larger for att than for the other instances.

From the analysis of results, two other conclusions can be drawn. The path length is almost constant through the generations for all four instances tested (we do not present plots for this experiment). And, as we can see in the plots, the path lengths vary only slightly with the total demand.

### 6.3.2 Number of paths

This experiment measures the minimum, the maximum, and the average number of paths among all paths of all o-d demand pairs. The measure is calculated for int-DEFT and OSPF for the best solution of the 1000th generation, for four demand matrices of four instances. Two paths are considered different if one has at least one arc that the other does not have. Thus, a path is considered different from a set of paths if it is different of each

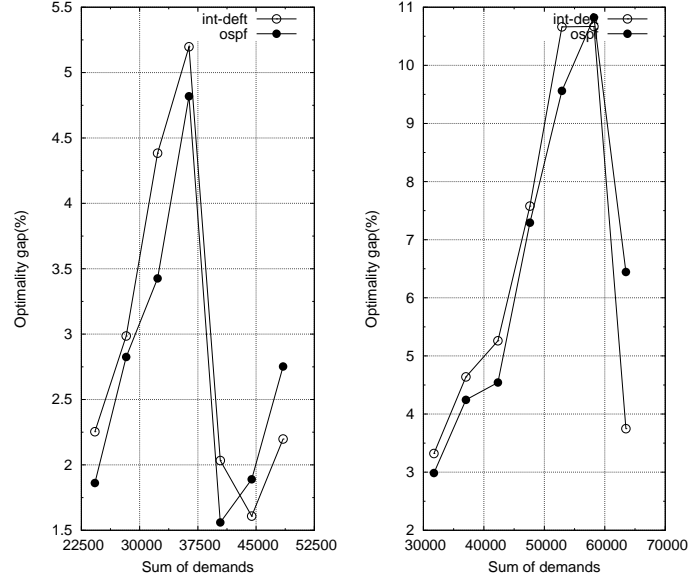


Figure 11: MA optimality gap for instances wax100 and wax100a measured for OSPF and DEFT solutions.

path of the set. Figure 13 presents the measures for the solution found by int-DEFT and OSPF.

The average number of paths found by int-DEFT is about 10 times higher than the average number found in OSPF solutions. For example, considering the demand matrix with the largest total demand, the average values for int-DEFT are 10.13, 12.67, 5.08, and 9.23, whereas the corresponding values for OSPF are 1.63, 1.77, 1.36, and 1.08.

The difference between the number of paths between int-DEFT and OSPF could be even larger if we use a larger threshold for  $h_{u,v}^t$ . We decided to use a threshold equal to nine to avoid having a very small amount of load for a demand pair flowing in a link.

From the figures, one can observe that the number of paths changes slightly with the total demand. Further experiments also showed that the number of paths is about the same over the generations.

### 6.3.3 Percentage of intermediate nodes affected

This experiment has the objective of showing the distribution of paths in the network, i.e., it presents the percentage of nodes that are part of some path used in an o-d demand pair. It measures the smallest, the largest, and the average percentage of intermediate nodes among all paths of all o-d demand pairs. The measure is calculated for int-DEFT and OSPF for the best solution of the 1000th generation, for four demand matrices of the four instances used in the experiments. Figure 14 depicts the number of intermediates nodes for int-DEFT and OSPF.

In the experiments performed, the percentage of intermediate nodes of int-DEFT is almost twice the percentage of intermediate nodes of OSPF. Since int-DEFT sends flows among all forward links, it is expected that a larger part of the graph would be used for routing a demand pair. As for the path length, the larger is the set of intermediate nodes, the higher is the probability of a demand pair is affected in the case of a link or node failure. For example, considering the demand matrix with the largest total demand, the average percentages of intermediate nodes used in the int-DEFT solution are 13.98, 12.23, 18.93, and 12.61, whereas the corresponding values for the OSPF solution are 8.61, 8.37, 11.83, and 4.57.

The minimum percentage of intermediate nodes is about the same for int-DEFT and OSPF. The maximum percentage for DEFT is about twice the maximum percentage found by OSPF in most instances. On instance wax100 the percentage of intermediate nodes in int-DEFT is three times the percentage of OSPF.

We also observed that the percentage of intermediate nodes is about the same throughout the generations of the algorithm. Finally, as can be observed in the figures, the values did not change much when different demand matrices were considered.

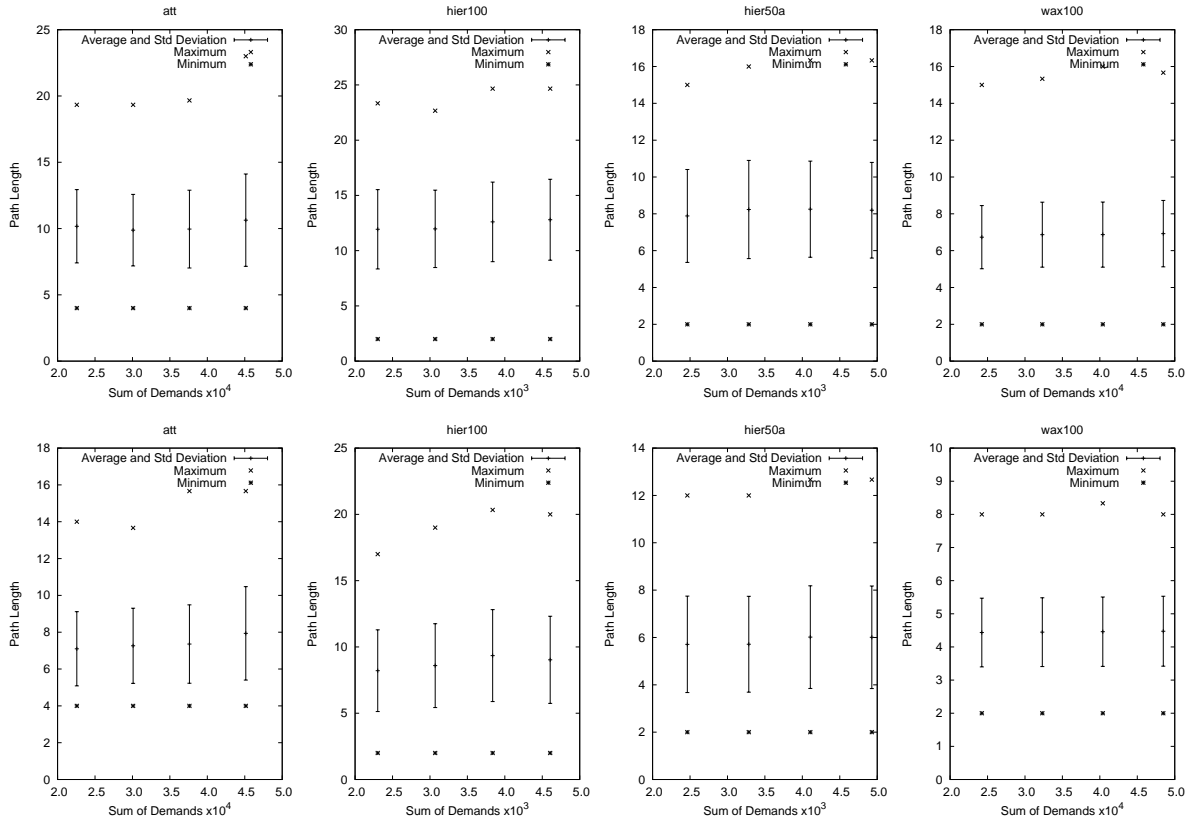


Figure 12: Path lengths for demands 6, 8, 10, and 12, and 1000 generations. Upper row: DEFT routing. Lower row: OSPF routing.

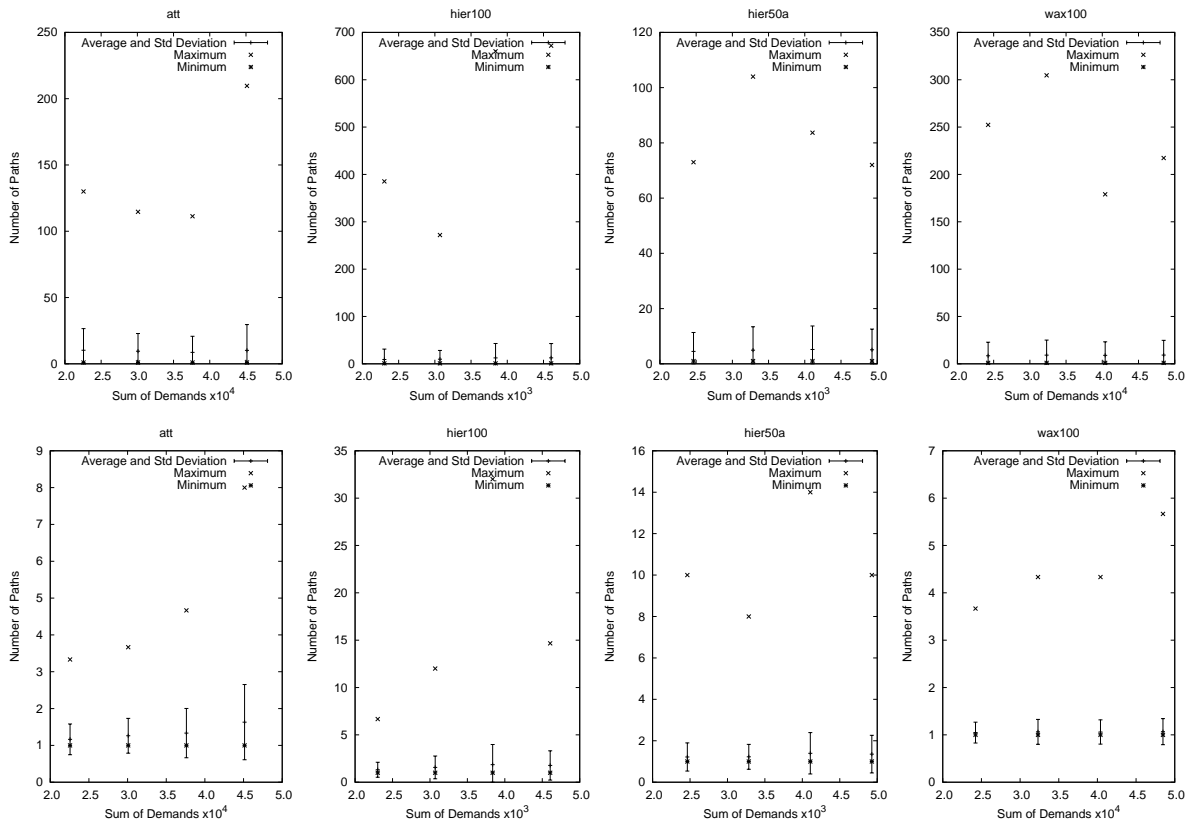


Figure 13: Number of paths for demands 6, 8, 10, and 12, and 1000 generations. Upper row: DEFT routing. Lower row: OSPF routing.

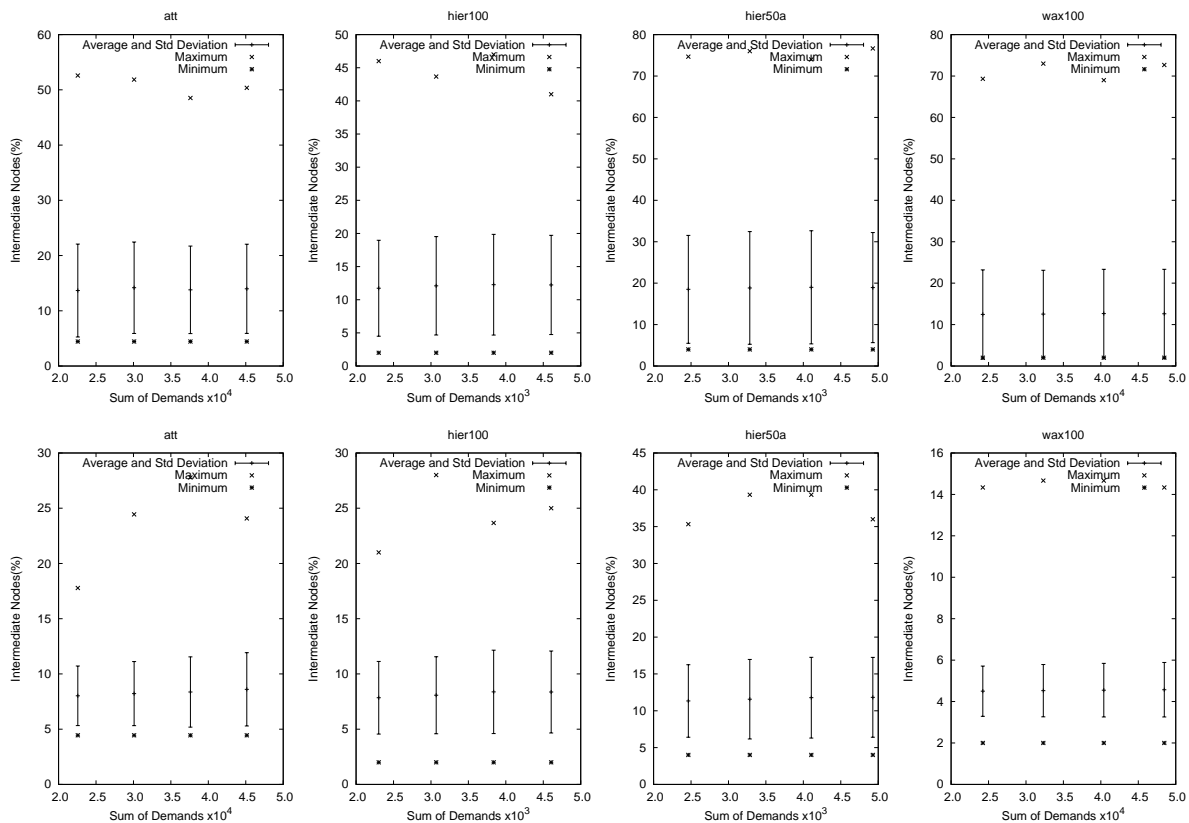


Figure 14: Percentage of intermediate nodes for demands 6, 8, 10, and 12, and 1000 generations for DEFT. Upper row: DEFT routing. Lower row: OSPF routing.

## 7 Conclusions and future work

In this paper, we presented a memetic algorithm applied to the weight setting problem for DEFT routing with integer weights. Our aim was to explore DEFT with integer weights to investigate whether it is comparable with OSPF.

A MA was previously applied for OSPF routing, and in this paper we compare these routing protocols. The MA uses dynamic shortest path and routing computation in both cases.

The results have shown that int-DEFT achieves better results than OSPF, i.e., int-DEFT routing produces less network congestion than OSPF routing, under the same available resources. However, int-DEFT produces solutions with longer path lengths, larger percentage of intermediate nodes, and larger number of paths. These are undesirable characteristics because under a link failure, a larger number of o-d demand pairs is expected to be affected.

In comparison with the two-stage method proposed for DEFT [15], our MA for int-DEFT finds solutions with higher network congestion. To make a fair comparison with that method, we intend modify the MA to allow for real weights in the network.

## Acknowledgments

We thank Dahai Xu for allowing us to experiment with his DEFT solver and for helpful comments.

## References

- [1] J.C. Bean. Genetic algorithms and random keys for sequencing and optimization. *ORSA J. on Computing*, 6:154–160, 1994.
- [2] W. Ben-Ameur and E. Gourdin. Internet routing and related topology issues. *SIAM J. on Discrete Mathematics*, 17:18–49, 2003.

- [3] A. Bley. *Integer Programming and Combinatorial Optimization*, volume 3509/2005 of *Lecture Notes in Computer Science*, chapter On the Approximability of the Minimum Congestion Unsplittable Shortest Path Routing Problem, pages 97–110. Springer Berlin / Heidelberg, 2005.
- [4] P. Broström and K. Holmberg. Multiobjective design of survivable IP networks. *Annals of Operations Research*, 147:235–253, 2006.
- [5] L.S. Buriol, M.G.C. 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.
- [6] M. Ericsson, M.G.C. Resende, and P. Pardalos. A genetic algorithm for the weight setting problem in OSPF routing. *J. of Combinatorial Optimization*, 6:299–333, 2002.
- [7] J.H. Fong, A.C. Gilbert, S. Kannan, and M.J. Strauss. Better alternatives to OSPF routing. *Algorithmica*, 43(1-2):113–131, 2005.
- [8] B. Fortz and M. Thorup. Internet traffic engineering by optimizing OSPF weights. In *INFOCOM 2000*, pages 519–528, 2000.
- [9] B. Fortz and M. Thorup. Increasing internet capacity using local search. *Computational Optimization and Applications*, 29(1), 2004.
- [10] K. Holmberg and D. Yuan. Optimization of internet protocol network design and routing. *Networks*, 43:39–53, 2004.
- [11] K.A. De Jong and W.M. Spears. An analysis of the interacting roles of population size and crossover. In *In Proceedings of the Int’l Workshop Parallel Problem Solving from Nature*, pages 38–47, 1990.
- [12] P. Moscato. On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms. Technical Report C3P Report 826, Caltech Concurrent Computation Program, 1989.
- [13] G. Ramalingam and T. Reps. An incremental algorithm for a generalization of the shortest-path problem. *J Algorithms*, 21:267–305, 1996.
- [14] N. Wang, K. Ho, G. Pavlou, and M. Howarth. An overview of routing optimization for internet traffic engineering. *IEEE Communications Surveys & Tutorials*, 10(1):36–56, 2008.
- [15] D. Xu, M. Chiang, and J. Rexford. DEFT: Distributed exponentially-weighted flow splitting. In *Proc. 26th IEEE Conf. on Computer Communicatios (INFOCOM)*, pages 71–79, May 2007.
- [16] D. Xu, M. Chiang, and J. Rexford. Link-state routing with hop-by-hop forwarding can achieve optimal traffic engineering. In *Proc. 27th IEEE Conf. on Computer Communicatios (INFOCOM)*, pages 466–474, 2008.