A BIASED RANDOM-KEY GENETIC ALGORITHM FOR OSPF AND DEFT ROUTING TO MINIMIZE NETWORK CONGESTION

ROGER REIS, MARCUS RITT, LUCIANA S. BURIOL, AND MAURICIO G.C. RESENDE

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 paper, we present a biased random-key genetic 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, however, yielding larger delays.

1. INTRODUCTION

The Internet consists 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 can be represented as 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 within 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). This routing model makes the best possible use of 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 determining proper weights to

Date: Janaury 9, 2009. Revised December 15, 2009.

Key words and phrases. Networks, networking, telecommunication networks, routing, genetic algorithm, biased random-key genetic algorithm, local search, metaheuristics.

AT&T Labs Research Technical Report.

optimize an objective function or multiple conflicting objectives 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. The traffic constraints can include QoS (Quality of Service) features like delay, jitter, number of hops or cost. With these aims, 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 objective is to determine link weights such that if the traffic is routed according to the protocol, then congestion is minimized. This single-objective routing, however, has its limitations. For example, it does not deal directly with QoS requirements that one may also want to optimize. Multi-objective routing can help us address these requirements. See, e.g. Girão-Silva et al. (2009).

The weight setting problem in OSPF routing was proved to be NP-hard by Fortz and Thorup (2000). 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 biased random-key genetic algorithm (BRKGA) introduced by Ericsson et al. (2002), was applied on the same set of instances considered in Fortz and Thorup (2000), and also produced large gaps for the hardest instances. Buriol et al. (2005) incorporated a local search procedure into a modified version of the decoder of the BRKGA of Ericsson et al., resulting in a memetic algorithm (Moscato, 2002). 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 (2004). Subsequently, survivability aspects were introduced by Broström and Holmberg (2006).

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 (Ben-Ameur and Gourdin, 2003; Bley, 2005).

DEFT (*Distributed Exponentially-Weighted Flow SpliTting*) is an IGP routing protocol, recently proposed by Xu et al. (2007). DEFT considers not only shortest paths for routing. It directs flow through all forward paths, but with exponential costs for longer paths. Furthermore, DEFT weights are real numbers. The protocol was designed to simplify intradomain traffic engineering. Subsequently, Xu et al. (2008) 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 forwarding links. In PEFT, of an outgoing shortest path link belongs to more than one shortest path, 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. (2008) presents more than one hundred up-to-date references on different aspects of optimization in Internet routing.

This paper compares the algorithm presented in Buriol et al. (2005) 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 (1996), whereas to the best of our knowledge, dynamic OSPF routing update was first proposed by Fortz and Thorup (2000). 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. (2007). 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. The biased random-key genetic algorithm (BRKGA) framework for combinatorial optimization is reviewed in Section 4. Sections 5 and 6 describes customizations of the BRKGA for, respectively, the OSPF and DEFT weight setting problems. Computational results are presented in Section 7 and concluding remarks are made in Section 8.

2. THE GENERAL ROUTING PROBLEM

Let $G = (V, E)$ be a directed graph modeling a network with a set of 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. $T = \{v \in V : \ D_{uv} > 0, \ u \in V\}$. The *general routing problem* is to find the flows $f_{u,v}$ on each arc $(u, v) \in E$ such that an appropriate objective function is minimized and all demands are delivered from their source nodes to their target nodes. The objective function to be minimized is

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

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

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

proposed in Fortz and Thorup (2000; 2004) and shown in Figure 1.
Let $f^t_{u,v}$ be the flow with destination node t on link (u, v). Then, at all intermediate nodes $v \neq t$ any resulting flow must respect flow conservation constraints

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

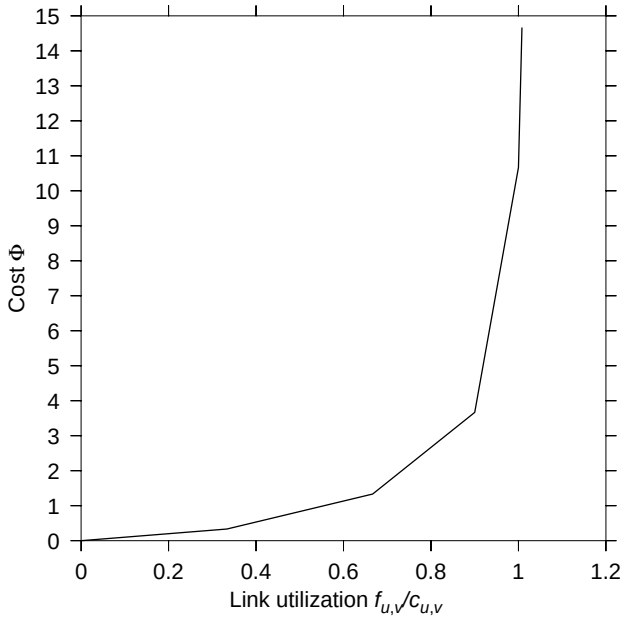

FIGURE 1. Link cost Φ as a function of the link utilization $f_{u,v}/c_{u,v}$.

and the individual flow aggregation

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

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), (2), (3), and the non-negativity constraints $f^t_{u,v} \geq 0$ and $f_{u,v} \geq 0$, for all $(u, v) \in E$ and $t \in T$. 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 up-to-date. 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 restriction 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 carry less flow. Formally, let d^t_u be the distance from node u to destination node t. Then $h^t_{u,v} = d^t_v + w_{u,v} - d^t_u$ 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 Γ in the direction to target node t traversing link (u, v) is defined as

$$\Gamma(h^t_{u,v}) = \begin{cases} e^{-h^t_{u,v}} & \text{if } d^t_u > d^t_v \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

and the fraction of the total flow $\Gamma(h^t_{u,v}) / \sum_{w:(u,w)\in E} \Gamma(h^t_{u,w})$ is calculated for each outgoing link (u, v) of u. According to Xu et al. (2007), 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 between the link weight setting for OSPF and DEFT, consider the example in Figure 2. The figure shows a network with a link weight increase in arc (b, t) from weight P_2 (left) to weight P'_2 (right). Suppose $P_1 > P'_2 > P_2 > 0$ 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 fraction of the traffic and the change in the weight of arc (b, t) causes a change 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.

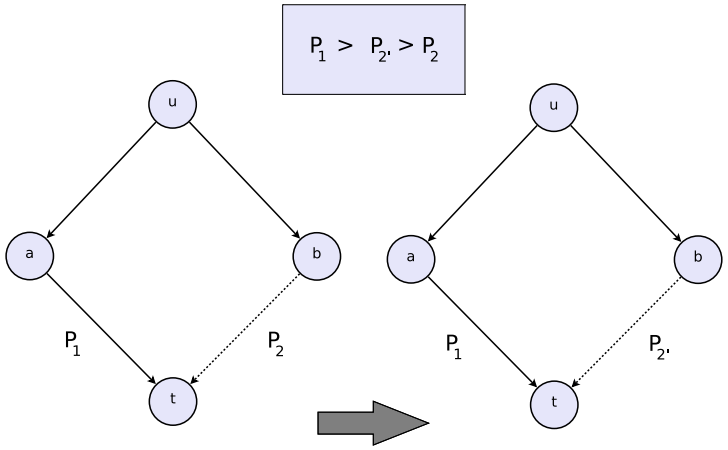

FIGURE 2. Effect in OSPF and DEFT routing when increasing the weight of arc (b, t).

The WSP for OSPF was formulated in Fortz and Thorup (2000) and different heuristics, e.g. (Buriol et al., 2005; Ericsson et al., 2002), have been proposed for its solution. DEFT was proposed in Xu et al. (2007), and to the best of our knowledge, no heuristic other than the biased random-key genetic algorithm described in this paper has been proposed to find optimal or near optimal solutions of the WSP for DEFT. A two-stage iterative algorithm for DEFT with real-valued weights based on non-linear and non-smooth optimization was proposed in Xu et al. (2007).

4. BIASED RANDOM-KEY GENETIC ALGORITHMS

Genetic algorithms, or GAs, (Goldberg, 1989; Holland, 1975) mimic *survival of the fittest* to find good-quality (optimal or near-optimal) solutions to combinatorial optimization problems. Solutions are associated with *individuals* in a *population*. Each individual's

chromosome encodes the solution. Each chromosome is made up of of strings of *genes*, each of which takes on a value, called an *allele*, from some alphabet. The *fitness* of a chromosome is correlated with the objective function value of the solution encoded by the chromosome. Over a number of *generations*, individuals that make up a population are evolved. At each generation, offspring of the current population are produced to make up the population of the next generation. Mutation takes place in genetic algorithms as a means to escape entrapment in local minima. Individuals are selected at random for mating. The probability that an individual is selected is proportional to the fitness of that individual. This way, the genetic material from the most fit individuals is passed on to the next generation (survival of the fittest).

Bean (1994) introduced genetic algorithms with random keys (RKGAs) for solving combinatorial optimization problems involving sequencing. In a RKGA, chromosomes are represented as a vector of random real numbers (random keys) in the interval $[0, 1]$. A *decoder* is a deterministic algorithm that takes as input any vector of random keys and associates with it a solution of the combinatorial optimization problem for which an objective value or fitness can be computed. For the sequencing problems addressed in Bean (1994), the decoder sorts the vector of random keys and uses the indices of the sorted keys to represent a sequence.

The initial population in a RKGA is made up of P vectors of random-keys. Each key is generated at random in the real interval $[0, 1]$. After decoding each individual, the population is partitioned into two groups of individuals: a smaller group of p_e *elite* individuals, i.e. those with the best fitness values, and a larger set with the remaining $P - p_e$ *non-elite* individuals, where $p_e < P - p_e$. To evolve the population, a new generation of individuals must be produced. An RKGA uses an *elitist strategy* to evolve the population from one generation to the next. In such a strategy, all of the elite individuals of generation k are copied unchanged to generation $k + 1$. RKGAs implement mutation by introducing *mutants* into the population. A mutant is simply a vector of random keys generated in the same way that an element of the initial population is generated. At each generation a small set of p_m mutants is introduced into the population. Discounting the the p_e elite individuals and the p_m mutants, $P - p_e - p_m$ additional individuals need to be produced to complete the P individuals that make up the population of the next generation. These offspring are produced through mating.

The evolutionary dynamics of a RKGA is described next. After all individuals are sorted by their fitness values, the population is partitioned into a set of ELITE solutions, containing most fit solutions, and another of the remaining NON-ELITE solutions. The p_e elite random-key vectors are copied without change to the next population. The p_m mutant individuals are randomly generated and placed in the new population. The remainder of the population of the next generation is completed by crossover. In a RKGA, Bean (1994) selects two parents to mate at random from the entire population. A *biased random-key genetic algorithm,* or BRKGA (Gonçalves and Resende, 2009), is similar to a RKGA except in the way parents are selected for mating. In a BRKGA, each offspring is generated by mating one parent chosen at random (with repetition) from the ELITE partition in the current population and one chosen at random (also with repetition) from the NON-ELITE partition. This way, an individual can produce more than one offspring in the same generation. Mating in both RKGA and BRKGA heuristics is done with *parameterized uniform crossover* (Spears and DeJong, 1991). Let $\rho_e > 0.5$ denote the probability that an offspring inherits the key of its elite parent and let n denote the number of keys in a random-key vector. For $i = 1, \ldots, n$, the i-th allele $c(i)$ of the offspring c takes on the value of the i-th

allele $e(i)$ of the elite parent e with probability ρ_e and the value of the i-th allele $\bar{e}(i)$ of the non-elite parent $\bar{e}$ with probability $1 - \rho_e$. This way, an offspring is more likely to inherit characteristics of the elite parent than those of the non-elite parent (survival of the fittest). Since we assume that any random key vector can be decoded into a solution, then the offspring resulting from mating is always valid, i.e. can be decoded into a solution of the combinatorial optimization problem.

To define a BRKGA, one only needs to define how solutions are encoded and decoded. In the next two sections, we define BRKGAs for the weight setting problems in OSPF and DEFT routing.

5. A BIASED RANDOM-KEY GENETIC ALGORITHM FOR OSPF ROUTING

A BRKGA for the weight setting problem (WSP) in OSPF routing was introduced by Ericsson et al. (2002) and further developed by Buriol et al. (2005). Using parameter settings proposed by the above papers, we partition the population such that the set of elite solutions is made up of the 25% most fit individuals ($p_e = .25p$). Likewise, the number of mutants created every generation consists of 5% of the new population ($p_m = .05p$). In addition to the standard BRKGA described in Section 4, another form of mutation is used in our heuristics for the weight setting problem. For each gene, with a probability of 1%, each allele of the child inherits, in the crossover operator, a new random key in the interval $(0, 1)$ If the child does not inherit the new random key, then the probability that a child inherits the allele of the elite parent is 70% ($\rho_e = .7$). In the remainder of this section, we describe the decoder.

Given a vector of random keys, the decoder produces a network flow for which the congestion is computed with the network congestion function (1). An individual is encoded as a vector x of $n = |E|$ random keys where each random key $x_i \in (0, 1)$, for $i = 1, \ldots, n$. Given, x, an initial weight vector is defined as $w_i = \lceil x_i \times w_{\max} \rceil$, where $w_{\max} = 20$. This way, initial edge weights are integers in the interval $[1, 20]$. Starting from this weight vector, a fast local search is applied to try to decrease network congestion by simple changes in individual edge weights.

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 $w_{\max}$, 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 (within a given range) 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 weights increased without improving the solution. We tested different values of k and decided to set $k = 5$, as in the original procedure proposed in Buriol et al. (2005) 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.

6. A BIASED RANDOM-KEY GENETIC ALGORITHM FOR DEFT ROUTING

In this section we present the above described BRKGA modified for DEFT. We maintained the entire BRKGA structure as well as the operators described in the previous section, only changing the routing procedure. Thus, in this section we mention the modifications to the decoder when routing is done with DEFT, whereas in Subsection 6.1 the dynamic flow computation for DEFT is detailed.

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.

As 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 that is considerably longer than the shortest path to the target will receive a flow. This flow, however, can be very small, 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 Xu et al. (2007, section II.C). Our implementation works with integer weights, but uses a scaling parameter p. Actual real-valued distances are obtained by dividing the integer distances by p. We call our implementation int-DEFT when 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 (4) is modified to

$$\Gamma(h_{u,v}^t) = \begin{cases} e^{-h_{u,v}^t/p} & \text{if } d_u^t > d_v^t \text{ and } h_{u,v}^t \leq g \\ 0 & \text{otherwise} \end{cases}. \tag{5}$$

Figure 3 describes in pseudo-code 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 (Γ_{total}) of the flow distribution function (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. In line 14, we calculate the total demand f (traversing and leaving the current node) per unit of Γ. 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 Γ. 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.

procedure CostEvalDEFT($G = (V, E)$,T,D,$(w_e)_{e\in E}$, g)
for $\forall\, t \in T$
  d =ReverseDijkstra(t,w)
  G^t = ComputeShortestPathGraph(G,w,d)
  [ComputePartialLoads(d, G^t, D)]
    H = sorted nodes in decreasing order of distances
    for each $u \in H$
      Γ_{total} = 0
      for each $v \in OUT(u)$
        if $d^t_u > d^t_v$ and $h^t_{u,v} \leq g$ **then**
          $h^t_{u,v} = d^t_v + w_{u,v} - d^t_u$
          $\Gamma_{total} = \Gamma_{total} + e^{-h^t_{u,v}}$
        endif
      endfor
      $f = (D_{u,t} + \sum_{v:(v,u)\in G^t} f^t_{v,u})/\Gamma_{total}$
      for each $v \in OUT(u)$
        if $d^t_u > d^t_v$ and $h^t_{u,v} \leq g$ **then**
          $\gamma = e^{-h^t_{u,v}}$
          $f^t_{u,v} = f\gamma$
        endif
      endfor
    endfor
  [end of ComputePartialLoads]
  for each $(u, v) \in E$
    $f_{u,v} = f_{u,v} + f^t_{u,v}$
endfor
$\Phi = \sum_{(u,v)\in E} \phi(u, v)$
end CostEvalDEFT.

FIGURE 3. Pseudo-code of solution evaluation in DEFT.

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.

6.1. **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 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 Δ 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 Δ. Second, nodes belonging to paths forwarding to t, in the case that before and after the change all forwarding paths traverse arc (u, v), and whose distances have changed, have no modifications in their loads. The loads traversing these nodes are affected equally by any Δ variation of the arc weight. Therefore, the flow distribution is unaltered. In the third case are those nodes that, with increment Δ, 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 Δ in arc (u, v), nodes u and a, that are above the dashed curve, 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 (b, v), while links (a, u) and (u, v) have their loads decreased.

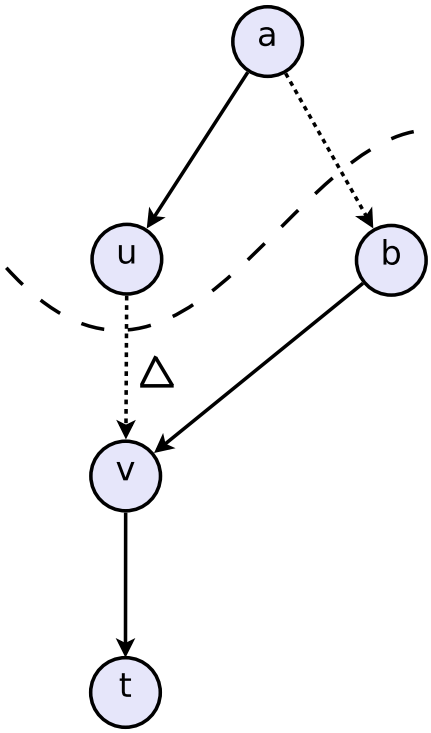

FIGURE 4. Dynamic flow update.

Figure 5 presents the pseudo-code 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 (*condition2*) and a different successor whose shortest distance to t is increased by exactly Δ (*condition1*). 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 Δ increase) to t. The heap is ordered by the current distance and, in case of a tie, by the previous distance. More formally,

$$(d, d_{old}) < (d', d'_{old}) \quad \text{if} \quad d < d' \text{ or } (d = d' \text{ and } d_{old} < d'_{old}). \tag{6}$$

If two nodes have the same distance, the node with the smaller distance before the update (d_{old}) has to be processed first, to avoid evaluating links whose flow was not yet updated. With the order establish by Equation (6) the flow on links is reset (line 33 of the algorithm) before being calculated, guaranteeing to not consider the old flows on those links.

The loop in lines 14–18 adds to the heap every intermediate node that is a successor in the direction of t of a node whose flow was altered. The new flow distribution is calculated in lines 19–37. Line 27 calculates the total flow leaving the current node. In lines 28–35, 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 Φ 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.

7. EXPERIMENTAL RESULTS

We have studied the performance of the BRKGA 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 Fortz and Thorup (2004) and Xu et al. (2007) 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 into four groups: 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 Fortz and Thorup (2000) and Buriol et al. (2005).

TABLE 1. Instances used in computational experiments.

Instance	Name	Nodes	Links	Capacities	O-D pairs
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

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

FIGURE 5. Pseudo-code of the dynamic flow algorithm.

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 BRKGA was tested with the following parameters:

TABLE 2. Speedup of int-DEFT-opt and int-DEFT over int-DEFT-DSSSP.

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
	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	

- Population size: $P = 50$ individuals,
- Weight interval: $[1, w_{\max}] = [1, 20]$,
- Algorithm running time: 60 minutes,
- Probability of inheriting allele from elite parent during crossover: $\rho_e = 0.7$,
- Maximum gap for $h^t_{u,v}$: $g = 9$,
- Scaling 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 (Buriol et al., 2005).

All experiments were carried out 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 analyzes 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 the BRKGA for each combination of network and total demand.

7.1. **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 6.1 compared to a straightforward adaptation of the original algorithm proposed for OSPF. We compare three different implementations:

int-DEFT-DSSSP: A straightforward adaptation to DEFT of the BRKGA 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-DSSSP, with its data structures tuned to work efficiently with DEFT, decreasing the running times. In particular, the data structure that maintains the number of links leaving each node that belongs to a shortest path to the destination node (in OSPF) is unnecessary in DEFT.

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

It is important to mention that the solution is always the same, independent of the implementation. However, the running times are affected by the implementation being used. 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-DSSSP. All 13 instances were tested with the highest total demand.

The straightforward adaptation from OSPF is on 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 due to the dynamic flow computation. Tuning the implementation for DEFT results only in an

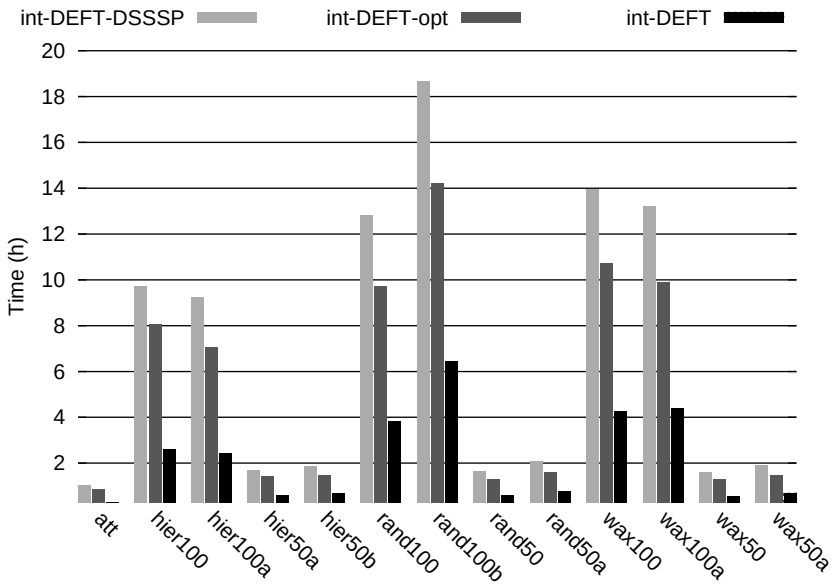

FIGURE 6. Running times (in hours) of 1000 generations for three DEFT routing implementations.

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 DEFT it is worthwhile to implement dynamic flow computation.

7.2. **Quality of solutions.** The second set of experiments compares the quality of the solutions obtained by the BRKGA 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, considering 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.

7.3. **Analysis of the routing paths.** The third experiment we performed aims to analyze 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 for 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.

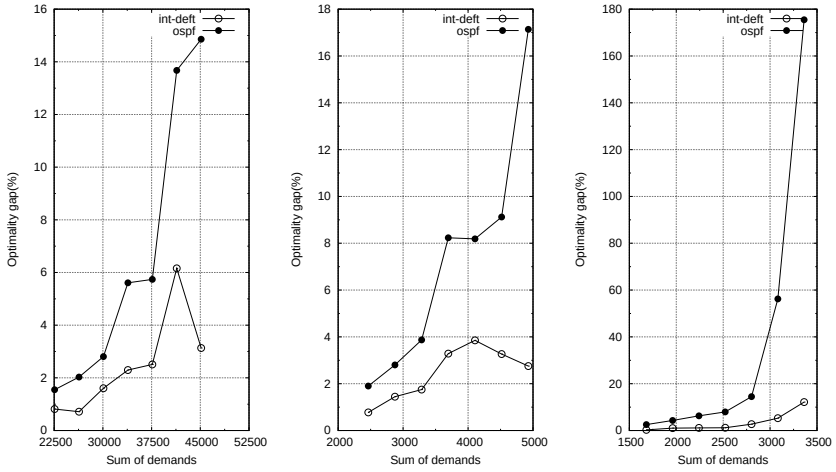

FIGURE 7. BRKGA optimality gap for instances `att`, `hier50a`, and `hier50b` measured for OSPF and DEFT solutions.

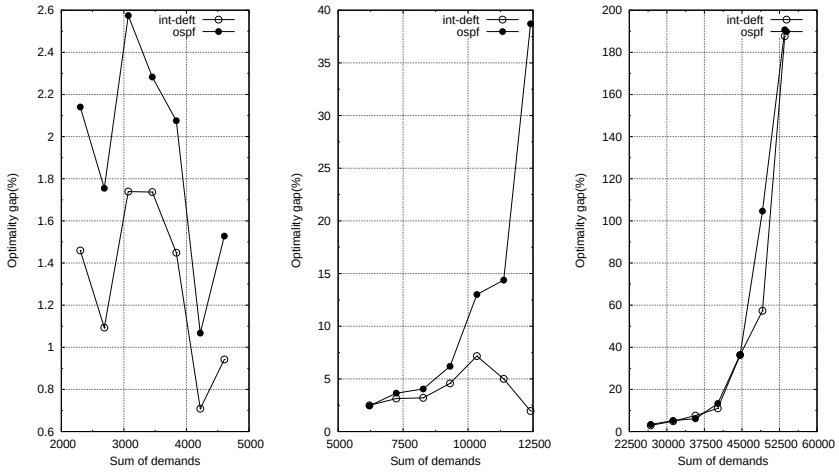

FIGURE 8. BRKGA optimality gap for instances `hier100`, `hier100a`, and `rand50a` measured for OSPF and DEFT solutions.

(3) **Percentage of intermediate nodes affected**: the average number of intermediate nodes routing an 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.

7.3.1. *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

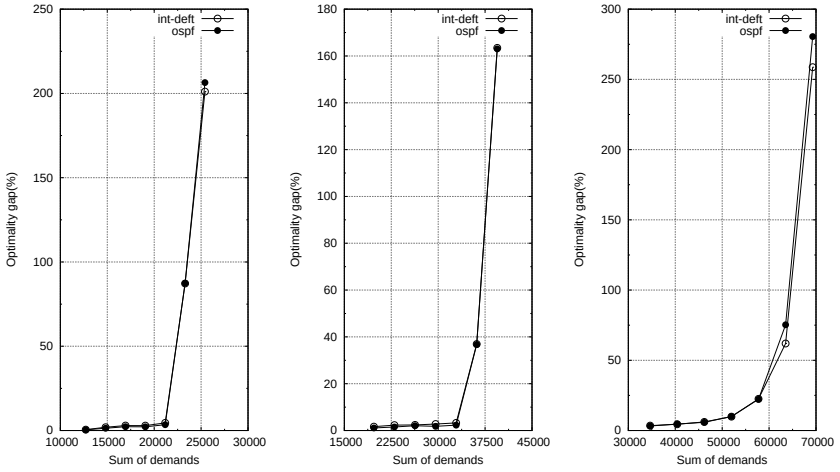

FIGURE 9. BRKGA optimality gap for instances `wax50`, `wax50a` and `rand100` measured for OSPF and DEFT solutions.

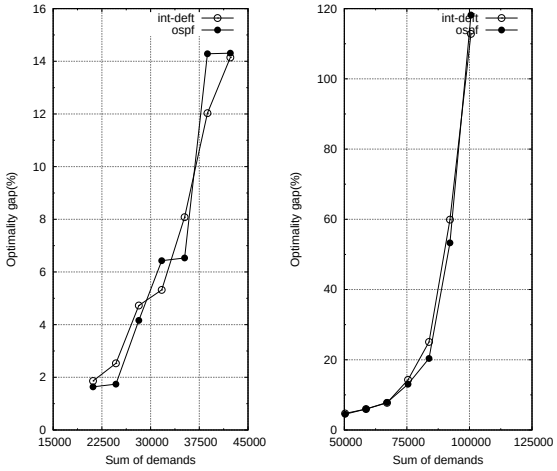

FIGURE 10. BRKGA optimality gap for instances `rand50` and `rand100b` measured for OSPF and DEFT solutions.

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 1000th 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 on 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 as the path length increases, so does the expected number of demand pairs affected

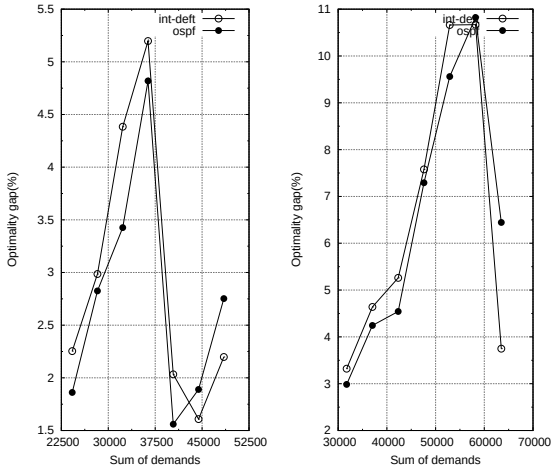

FIGURE 11. BRKGA optimality gap for instances `wax100` and `wax100a` measured for OSPF and DEFT solutions.

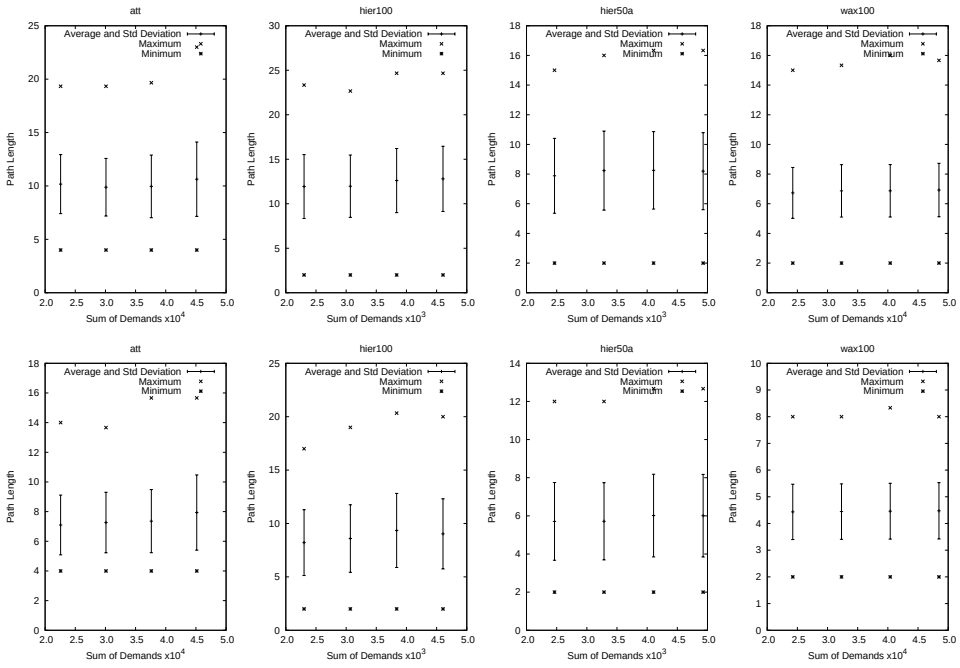

FIGURE 12. Average path length per O-D pair for demands 6, 8, 10, and 12, and 1000 generations. Upper row: DEFT routing. Lower row: OSPF routing.

by a failure. Thus, the length of a path is directly related to the quality of service of a telecommunication network.

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.

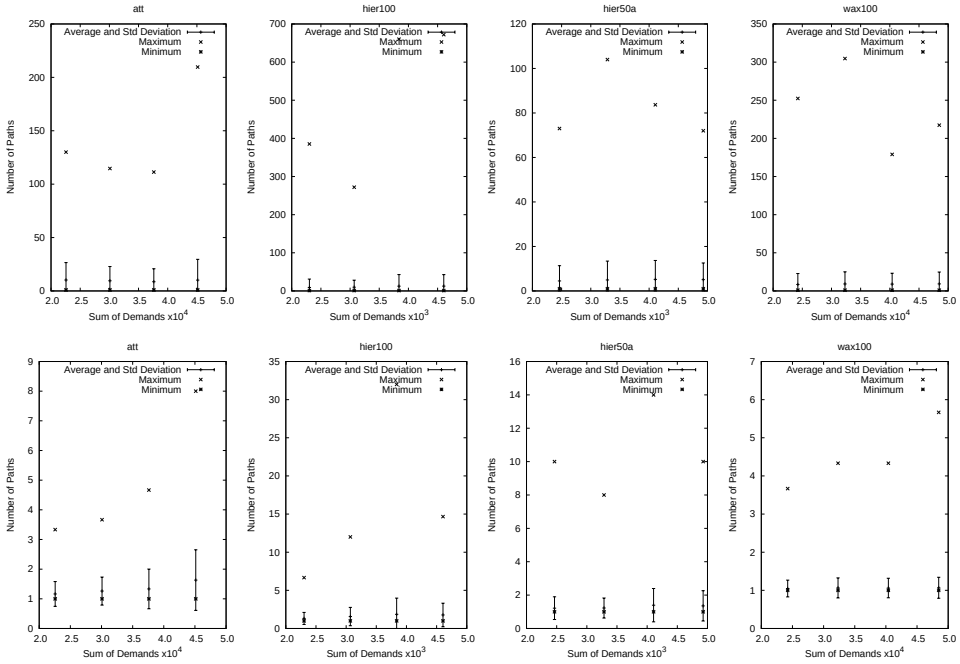

FIGURE 13. Average number of paths per O-D pair for demands 6, 8, 10, and 12, and 1000 generations. Upper row: DEFT routing. Lower row: OSPF routing.

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` does not have O-D demand pairs between all pairs of nodes, while the other instances do. 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). Finally, as we can see in the plots, the path lengths vary only slightly with the total demand.

7.3.2. *Number of paths.* This experiment measures the minimum, maximum, and 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 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 $h^t_{u,v}$. 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.

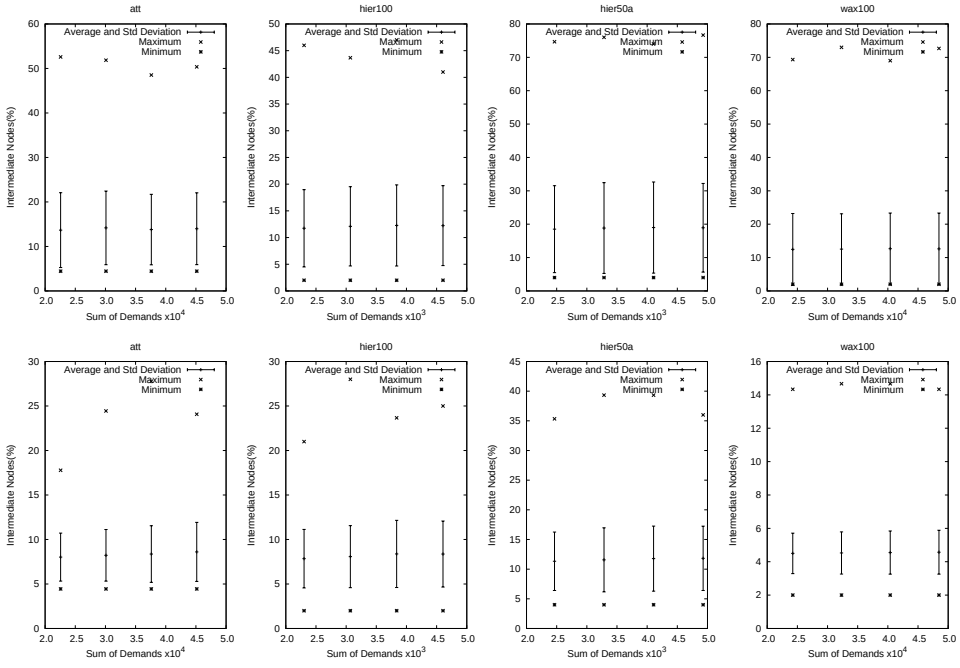

FIGURE 14. Percentage of intermediate nodes for demands 6, 8, 10, and 12, and 1000 generations for DEFT. Upper row: DEFT routing. Lower row: OSPF routing.

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.

7.3.3. *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, largest, and 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 the set of intermediate nodes, the higher the probability of a demand pair being 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.

8. Concluding remarks

This paper presented a biased random-key genetic algorithm (BRKGA) for the weight setting problem in DEFT routing with integer weights. Our main goal was to show that the BRKGA originally proposed by Ericsson et al. (2002) and Buriol et al. (2005) for the weight setting problem in OSPF routing could be adapted for the integer weight setting problem in DEFT routing (int-DEFT).

We compared results produced for weight setting in OSPF and int-DEFT routing. The results show that, using identical available resources, int-DEFT produces less network congestion than OSPF routing does. However, int-DEFT produces solutions with longer path lengths, larger percentage of intermediate nodes, and larger number of paths. These are all undesirable characteristics since a link failure will result in a larger expected number of affected O-D demand pairs.

In comparison with the two-stage method proposed for DEFT (Xu et al., 2007), the algorithm for int-DEFT routing, proposed in this paper, finds solutions with higher network congestion. A direct comparison requires either that we extend the BRKGA proposed in this paper to handle real-valued weights or adapt the two-stage method to handle integer weights.

A natural next step for this work is to extend the single-objective nature of the problem to one that is multi-objective. One may require, for example, that two conflicting objectives, such as the minimization of congestion and the minimization of delay be simultaneously optimized. Since genetic algorithms are commonly used for multi-objective optimization (Konak et al., 2006), biased random-key genetic algorithms should be well suited to address these multi-objective routing problems.

Acknowledgments

The authors would like to thank Dahai Xu for allowing us to experiment with his DEFT solver and for helpful comments.

References

J. C. Bean. Genetic algorithms and random keys for sequencing and optimization. *ORSA J. on Computing*, 6:154–160, 1994.

W. Ben-Ameur and E. Gourdin. Internet routing and related topology issues. *SIAM J. on Discrete Mathematics*, 17:18–49, 2003.

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.

P. Broström and K. Holmberg. Multiobjective design of survivable IP networks. *Annals of Operations Research*, 147:235–253, 2006.

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.

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.

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

B. Fortz and M. Thorup. Increasing internet capacity using local search. *Computational Optimization and Applications*, 29(1), 2004.

R. Girão-Silva, J. Craveirinha, and J. Clímaco. Hierarchical multiobjective routing in Multiprotocol Label Switching networks with two service classes: A heuristic solution. *International Transactions in Operational Research*, 16:275–305, 2009.

D.E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, 1989.

J.F. Gonçalves and M.G.C. Resende. Biased random-key genetic algorithms for combinatorial optimization. Technical report, AT&T Labs Research, Florham Park, NJ 07932, 2009.

J.H. Holland. *Adaptation in Natural and Artificial Systems*. MIT Press, 1975.

K. Holmberg and D. Yuan. Optimization of internet protocol network design and routing. *Networks*, 43:39–53, 2004.

A. Konak, D.W. Coit, and A.E. Smith. Multi-objective optimization using genetic algorithms: A tutorial. *Reliability Engineering and System Safety*, 91:992–1007, 2006.

P. Moscato. Memetic algorithms. In P.M. Pardalos and M.G.C. Resende, editors, *Handbook of applied optimization*. Oxford University Press, 2002.

G. Ramalingam and T. Reps. An incremental algorithm for a generalization of the shortest-path problem. *J Algorithms*, 21:267–305, 1996.

W.M. Spears and K.A. DeJong. On the virtues of parameterized uniform crossover. In *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 230–236, 1991.

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.

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.

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.

(R. Reis) 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.
E-mail address: rsreis@inf.ufrgs.br

(M. Ritt) 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.
E-mail address: mrpritt@inf.ufrgs.br

(L.S. Buriol) 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.
E-mail address: buriol@inf.ufrgs.br

(M.G.C. Resende) ALGORITHMS AND OPTIMIZATION RESEARCH DEPARTMENT, AT&T LABS RESEARCH, 180 PARK AVENUE, ROOM C241, FLORHAM PARK, NJ 07932 USA.
E-mail address, M.G.C. Resende: mgcr@research.att.com