DEFT: Distributed Exponentially-weighted Flow Splitting

Dahai Xu
Dept. of EE, Princeton University
Email: dahaixu@princeton.edu

Mung Chiang
Dept. of EE, Princeton University
Email: chiangm@princeton.edu

Jennifer Rexford
Dept. of CS, Princeton University
Email: jrex@cs.princeton.edu

***Abstract*— Network operators control the flow of traffic through their networks by adapting the configuration of the underlying routing protocols. For example, they tune the integer link weights that interior gateway protocols like OSPF and IS-IS use to compute shortest paths. The resulting optimization problem—to find the best link weights for a given topology and traffic matrix—is computationally intractable even for the simplest objective functions, forcing the use of local-search techniques. The optimization problem is difficult in part because these protocols split traffic evenly along shortest paths, with no ability to adjust the splitting percentages or direct traffic on other paths. In this paper, we propose an extension to these protocols, called Distributed Exponentially-weighted Flow SpliTting (DEFT), where the routers can direct traffic on non-shortest paths, with an exponential penalty on longer paths. DEFT leads not only to an easier-to-solve optimization problem, but also to weight settings that provably perform no worse than OSPF and IS-IS. Furthermore, in our optimization problem, both link weights and flows of traffic are integrated as optimization variables into the formulation and jointly solved by a two-stage iterative method. Our novel formulation leads to a much more efficient way to identify good link weights than the local-search heuristics used for OSPF and IS-IS today. DEFT retains the simplicity of having routers compute paths based on configurable link weights, while approaching the performance of more complex routing protocols that can split traffic arbitrarily over any paths.**

Keywords: Interior gateway protocol, traffic engineering, routing, OSPF, network optimization, mathematical programming.

I. Introduction

A. Motivation

Managing a large IP network is immensely challenging, in large part because the existing protocols and mechanisms were not designed with management in mind. For example, the design of existing protocols and mechanisms typically induces optimization problems that are computationally intractable, forcing the use of local-search techniques to identify good parameter settings. In this paper, we argue for "Design for Optimizability": protocols should be designed with the resulting optimization problems in mind, with enough flexibility and optimizability provided in the first place so as to enable efficient and easy-to-operate solutions. In particular, we show how to extend existing link-state routing protocols for more effective traffic engineering [1] within a single Autonomous System (AS), such as a company, university campus, or Internet Service Provider (ISP).

Most large IP networks run Interior Gateway Protocols (IGPs) such as OSPF (Open Shortest Path First) or IS-IS (Intermediate System-Intermediate System) that select paths based on link weights. Routers use these protocols to exchange link weights and construct a complete view of the topology inside the AS. Then, each router computes shortest paths (where the length of a path is the sum of the weights on the links) and creates a table that controls the forwarding of each IP packet to the next hop in its route. When multiple shortest paths exist, a router typically splits traffic roughly evenly over each of the outgoing links along a shortest path to the destination. The link weights are configured by the network operators or automated management systems, through centralized computation, to satisfy traffic-engineering goals, such as minimizing the maximum link utilization or the sum of link cost [2]. We will use the sum of link cost as the primary comparison metric and the optimization objective. A typical link cost function of link utilization is illustrated in Fig. 1 to model retransmission delays caused by packet losses [3].

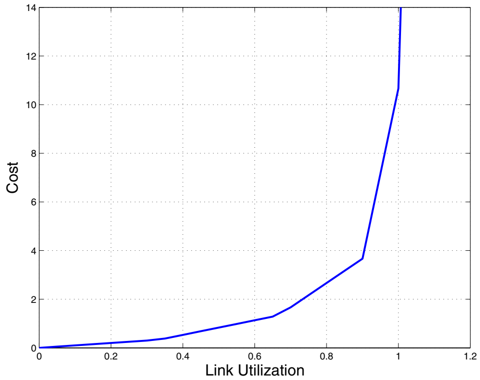

Fig. 1. Link cost as a function of the load for a unit link capacity

Tuning the link weights under OSPF and IS-IS[1] is a form of *link-weight-based* traffic engineering, where the link weights *uniquely* determine the flow of traffic in a *distributed* manner within the network for any given traffic matrix. The traffic matrix can be computed based on traffic measurements (e.g., [4]) or may represent explicit subscriptions or reservations from users. Link-weight-based traffic engineering has two key components: a *centralized* approach for setting the routing parameters (i.e., link weights) and a *distributed* way of using these link weights to compute the routes to forward packets. Setting the routing parameters based on a network-wide view of the topology and traffic, rather than the local views at each router, can achieve better performance [5].

[1] The integer link weight could be $1 \sim 2^{16} - 1$ for OSPF and $1 \sim 2^{6} - 1$ for IS-IS (or $1 \sim 2^{24} - 1$ for the new version). We use OSPF to represent OSPF and IS-IS thereafter.

Link-weight-based schemes are appealing alternatives to more complex load-sensitive routing protocols for several reasons [5]. Link-weight schemes are compatible with existing link-state routing protocols, and link weights are a concise form of configuration state, with one parameter on each unidirectional link. The weights have natural default values (e.g., inversely proportional to link capacity or proportional to propagation delay). If the topology changes, the routers can automatically compute new routes based on the current topology and link weights. In addition, the resulting routing protocols have low overhead and are intrinsically stable, since the routers do not adapt automatically to locally-constructed (and potentially out-of-date views) of the traffic. Finally, link weights offer a great deal of flexibility for controlling the flow of traffic; often, changing just one or two link weights is sufficient to alleviate congestion in the network.

Evaluation of various traffic engineering schemes, in terms of total link cost minimization, can be made against the performance benchmark of optimal routing (OPT), which can direct traffic along any paths in any proportion. OPT models an idealized routing scheme that can establish one or more explicit paths between every pair of nodes, and distribute an *arbitrary* amount of traffic on each of the paths.

It is easy to construct examples where OSPF with the best link weighting performs substantially (5000 times) worse than OPT in terms of minimizing the sum of link cost [2]. In addition, finding the best link weights under OSPF is NP-hard [2]. Although the best OSPF link weights can be found by solving an integer linear program (ILP) formulation (as shown in Appendix A), such an approach is impractical even for a mid-size network. Many heuristics, including local search [2] and simulated annealing [6], [7] have been proposed to search for the best link weights under OSPF. Among them, local-search technique is the most attractive method in finding a good setting of the link weights for large networks. Even though OSPF with a good setting of the weights performs within a few percent of OPT for some practical scenarios [2], [6], [7], there are still many realistic situations whereas the performance gap between OSPF and OPT could be significant even at low utilization [2], [8].

In summary, OSPF's failure to achieve optimal routing has two underlying causes:

1) The protocol limitation on even splitting of traffic across multiple shortest-path routes;
2) The computational intractability of finding the best link weights.

We present an example to illustrate these two effects. For the network in Fig. 2, which has 5 nodes and 8 bi-directional edges (for a total of 16 links), the link capacities are all 5 units, and the traffic demand between each node pair is randomly chosen from [0, 5] units. The objective value (in terms of the sum of link cost) of optimal routing is 379.86 units. In contrast, the objective value from using the Best OSPF (ILP) is 631.16 units (with an optimality gap of 66.2%) and that from Heuristic OSPF (Local Search) is 3615.29 units (with a performance gap of 851.7%).

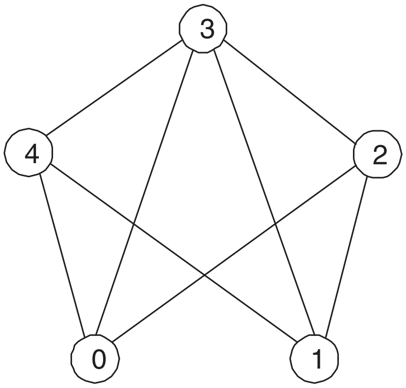

Fig. 2. An illustrative example to show Optimal Routing $>$ BEST OSPF (ILP) $>$ Heuristic OSPF (Local Search) in minimizing total link cost

Although OPT could be realized by some non-link-weight-based traffic engineering (e.g., [8]–[10]) each router cannot *independently* compute the flow splitting only based on link weights, and additional centralized signaling has to be implemented. Therefore, it is useful but challenging to search for a routing protocol with which the resulting link-weight-based traffic engineering can realize OPT.

B. Overview of DEFT

In light of the difficulty of tuning OSPF for consistently good performance, we wonder how close to optimal routing a link-state protocol could be. In this paper, inspired in part by Fong et al.'s work in [11], we explore the potential of link-weight-based traffic engineering by relaxing the constraint of shortest path routing as in OSPF. The proposed routing scheme is called Distributed Exponentially-weighted Flow SpliTting (DEFT), where the routers can direct traffic on non-shortest paths, with an exponential penalty on longer paths. DEFT's flexibility of routing on non-shortest paths can bring tremendous improvement in approaching network-wide traffic engineering objective and still keep the simplicity and scalability of link-state routing protocols. For the example scenario in Fig. 2, DEFT can achieve a flow with total link cost of 383.31 units (within 0.9% of optimality).

The second innovation of DEFT is a novel formulation of the optimization problem and a two-stage iterative solution method. Most existing methods of searching for good link weights under a link-state protocol, e.g., the local search OSPF in [3], start from a set of link weights that accordingly determine the flow of traffic, and tune the weights of some links to diversify the traffic. In this work, we develop an optimization formulation where both link weighting and traffic flows are variables at the same time, coupled through constraints in the formulation. Thus the solution to the formulation will bring an optimal link weighting at once and the searching procedure could be carried out much more efficiently. The detailed description of DEFT will be covered in Sec. II-III.

In the most relevant related work, Fong et al. [11] propose to forward traffic on paths in inverse proportion to (or strictly decreasing with) the sum of the weights. Accordingly, optimal routing for single-destination (sink) can be realized under the scheme within polynomial time. However, the approach may lead to loops in the routes, and its applicability and performance for the more crucial scenarios (with multiple-destinations) were not addressed.

C. Summary of Contributions

DEFT overcomes the two limitations of conventional OSPF traffic engineering because: (i) in the routing protocol, traffic can be routed on non-shortest paths and (ii) in the computation of link weights, the two-stage iterative method for optimizing the link weights is very effective. As a result, DEFT has the following desirable properties:

- It determines a unique flow of traffic for a given link weight setting in polynomial time.
- It is provably no worse than OSPF in terms of minimizing the maximum link utilization or the sum of link cost.
- It is readily implemented as an extension to the existing link-state protocols (e.g., OSPF).
- The two-stage iterative method realizes near-optimal flow of traffic even for large topologies.
- The optimization procedure for DEFT converges much faster than that for OSPF local search.

In summary, DEFT maintains the simplicity of link-state-based routing while attaining close-to-optimal congestion cost value through a link weight computation that runs as fast as today's OSPF local search.

The rest of the paper is organized as follows. We introduce the framework and prove the basic properties of DEFT in Sec. II, followed by the novel optimization formulation and its solution algorithms in Sec. III. Then we present results from extensive numerical experiments in Sec. IV, comparing DEFT with OSPF in terms of optimality gap, maximum link load, convergence behavior and complexity of the optimization procedure. We conclude and discuss future work on DEFT in Sec. V. Details of a reference optimization formulation and interior-point methods are outlined in the Appendix.

II. DEFT: Framework and Basic Properties

In this section, we introduce the framework and prove the basic properties of the DEFT routing scheme. Table I summarizes the key notation used throughout this paper.

A. Link-weight-based Traffic Engineering and DEFT

Given a directed graph $G = (\mathbb{V}, \mathbb{E})$ with capacity $c_{u,v}$ for each link (u,v), let $D(s,t)$ denote the traffic demand originated from node s and destined to node t. $\Phi(f_{u,v}, c_{u,v})$ is a strictly increasing convex function of flow $f_{u,v}$ on link (u,v) (typically a piece-wise linear cost [2], [8] as shown in equation (1), or in Fig. 1). The network-wide objective is to minimize $\sum_{(u,v)\in\mathbb{E}} \Phi(f_{u,v}, c_{u,v})$.

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

In link-weight-based traffic engineering, each router u needs to make an *independent* decision on how to split the traffic destined to node t among its outgoing links only using link weights. Therefore, it calls for a function ($\Gamma(\cdot) \geq 0$) to represent the traffic allocation.

In the case of shortest-path routing (e.g., OSPF), each router evenly splits flow across all the outgoing links as long as they are on shortest paths. First of all, we need a variable to indicate whether link (u,v) is on the shortest path to t or not. Denote $w_{u,v}$ as the weight for link (u,v), and d_u^t as the shortest distance from node u to node t, then $d_v^t + w_{u,v}$ is the distance from u to t when routed through v. Thus the gap of the two distances, $h_{u,v}^t \triangleq d_v^t + w_{u,v} - d_u^t$ is always greater than or equal to 0. Then (u,v) is on the shortest path to t if and only if $h_{u,v}^t = 0$. Accordingly, we can use a unit step function of $h_{u,v}^t$ to represent the traffic allocation for OSPF:

$$\Gamma(h_{u,v}^t) = \begin{cases} 1 & \text{if } h_{u,v}^t = 0 \\ 0 & \text{if } h_{u,v}^t > 0 \end{cases} \tag{2}$$

The flow proportion on the outgoing link (u,v) destined to t, at u is $\Gamma(h_{u,v}^t)/\sum_{(u,j)\in\mathbb{E}} \Gamma(h_{u,j}^t)$. Denote $f_{u,v}^t$ as the flow on link (u,v) destined to node t and f_u^t as the flow sent along the shortest path of node u destined to t, then

$$f_{u,v}^t = f_u^t\, \Gamma(h_{u,v}^t). \tag{3}$$

The $\Gamma(h_{u,v}^t)$ function (2) is in part responsible for the difficulty of optimizing the link weights under OSPF. For DEFT, we define a new $\Gamma(h_{u,v}^t)$ function to allow for flow on non-shortest paths. Intuitively, we should send more traffic on the shortest path than on a non-shortest path. Moreover, the traffic on a non-shortest path should be 0 if the distance gap between the non-shortest path and the shortest path is infinitely large. Based on the above intuition, $\Gamma(h_{u,v}^t)$ should be a strictly decreasing continuous function of $h_{u,v}^t$ bounded within $[0,1]$. The exponential function:

$$\Gamma(h_{u,v}^t) = \begin{cases} e^{-h_{u,v}^t} & \text{if } d_u^t > d_v^t \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

is one of the natural choices and the performance of using such function turns out to be excellent. From (4), we can easily verify that no packet would ever traverse a loop since the flow always makes forward progress towards the destination. In contrast, Fong et al. [11] propose to forward traffic on *all* paths in inverse proportion to (or exponentially decreasing with) path lengths. However, this approach may lead to loops in the routes, and its applicability and performance for routing with multiple destinations are not clear yet.

B. Sample Link Weighting in DEFT

To demonstrate the advantage of using DEFT over conventional OSPF, consider the example in Fig. 3 where all the traffic is travelling from node A to node B. The ratio of the traffic on the two paths with optimal routing could be $x : 1-x$ for any $0 \leq x \leq 1$ if the capacities on path $A \rightarrow 1 \rightarrow B$ and $A \rightarrow 2 \rightarrow B$ can be arbitrarily specified. On the other hand,

TABLE I

Summary of Key Notation

Notation	Meaning
$w_{u,v}$	Weight assigned to link (u, v)
w_{min}	Lower bound of all link weights
d_u^t	The shortest distance from node u to node t. $d_t^t = 0$
$h_{u,v}^t$	Gap of shortest distance, $h_{u,v}^t \triangleq d_v^t + w_{u,v} - d_u^t$
$\Gamma(h_{u,v}^t)$	Traffic allocation function
$f_{u,v}^t$	Flow on link (u, v) destined to node t
f_u^t	Flow along the shortest path of node u destined to t
$f_{u,v}$	Flow on link (u, v)
$c_{u,v}$	Capacity of link (u, v)
$D(s, t)$	Traffic demand from source s to destination t

such ratio under OSPF could only be $0 : 1$, $1 : 1$, or $1 : 0$. Therefore, to realize optimal routing, we have to send traffic along a non-shortest path.

For the example in Fig. 3, without loss of generality, path $A \rightarrow 1 \rightarrow B$ is assumed to be the shortest path with 1-unit length and its traffic fraction is $x \geq 0.5$. Therefore, we just need to assign $1{+}\log\frac{x}{1-x}$ units[2] as the length (weight) for path $A \rightarrow 2 \rightarrow B$, which will determine $1 - x$ traffic proportion on it under DEFT.

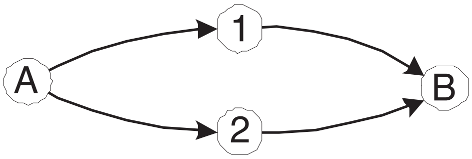

Fig. 3. A simple example of implementing optimal routing under DEFT

C. Realizing DEFT in Practice

The DEFT scheme can be easily implemented as a small extension to existing link-state routing protocols (like OSPF). First, the network operator or management system calculates the best link weights within DEFT for a given traffic matrix. Second, after receiving the updated link weights using link state advertisement (LSA) packets, each router independently determines the flow allocation across shortest and non-shortest paths to each destination according to (4). Thus the routing table stores several next hops (nodes) for each destination associated with the desired flow proportion. Such desired flow splitting can be approximately achieved by using pseudo-random methods (e.g., hashing the source and destination addresses and port number of the packet header [8], [12] to ensure that packets from the same TCP/UDP connection traverse the same path).

Although DEFT does not limit link weights to integer values, DEFT can also be efficiently implemented with integer weights. More specifically, assume the link weight for link (u, v) is set to $w_{u,v} \in [w_{min}, w_{max}]$ as the result of traffic engineering, we just to need to specify a global parameter, p, to convert $w_{u,v}$ into an integer weight by rounding $p\, w_{u,v}$. Let n be the number of bits to represent an integer weight in a routing protocol (e.g., $n = 16$ in OSPF), p could be specified as $\lfloor \frac{2^n - 1}{w_{max}} \rfloor$. For consistency, the rule of flow splitting in (4) can be replaced with (5) below.

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

If n is sufficiently large, the difference between using integer or non-integer link weights under DEFT is negligible. For $n = 16$ and all the scenarios tested in this work, the difference in terms of total link cost is usually less than 0.05% (with a single outlier of 0.4%).

Note that, by enabling the use of non-shortest paths, DEFT may direct some flows on paths with longer propagation delay. Fortunately, the exponential penalty in DEFT significantly limits the number of flows that traverse long paths. To tighten the bound on worst-case delay, the routers could limit the use of paths beyond a maximum target IGP distance. In general, most applications are not especially sensitive to delay, as long as delay stays below a target value. This allows DEFT to strike an attractive balance in achieving higher throughput than conventional link-state routing protocols, in exchange for a small increase in propagation delay for some flows.

D. Key properties

We prove the following key properties for DEFT.

***Theorem** 1: DEFT can realize any acyclic flow for a single-destination demand within polynomial time.*

Proof: The links without flow can be assigned infinitely large weights and excluded from further processing. Denote $f_u^t = \max_{(u,v)\in\mathbb{E}} f_{u,v}^t$, where $f_{u,v}^t$ is the amount of flow on link (u, v). The nodes are processed in their reverse topological order in the acyclic flow where the first node should be the destination t. When node u is processed, we set the shortest distance from node u to t, $d_u^t = \min_{(u,v)\in\mathbb{E}}(d_v^t - \log\frac{f_{u,v}^t}{f_u^t})$, and assign the weight of link (u, v) as $-\log\frac{f_{u,v}^t}{f_u^t} + d_u^t - d_v^t$. It is easy to verify that the above link weighting satisfies the definition of DEFT (4) [3]. ■

***Theorem** 2: DEFT can achieve optimal routing with a single destination within polynomial time.*

Proof: The optimal routing for a strictly increasing convex cost function can be achieved within polynomial time since all the constraints are linear and the resulting formulation (see Appendix A) is a convex optimization problem [3]. Obviously, such optimal flow for single destination is acyclic. From Theorem 1, such optimal flow can be realized using DEFT. ■

Note that, in contrast, OSPF cannot even realize optimal single destination flow for some scenarios [2] including the simple example (Fig. 3) introduced in Sec. II-B.

[2] It is derived from $\frac{x}{1-x} = \frac{\Gamma(h_{A\rightarrow 1\rightarrow B})}{\Gamma(h_{A\rightarrow 2\rightarrow B})} = \frac{e^0}{e^{-(w_{A\rightarrow 2\rightarrow B} - 1)}}$ where $w_{A\rightarrow 2\rightarrow B}$ is the weight for path $A \rightarrow 2 \rightarrow B$. Although $1 + \log\frac{x}{1-x}$ could be infinitely large when x reaches 1, a large enough weight assigned to path $A \rightarrow 2 \rightarrow B$ will make the traffic on the path negligible.

[3] All d_v^t have been determined since the nodes are processed in the reverse topological order and $d_t^t \equiv 0$

***Theorem** 3: DEFT is no worse than OSPF in terms of minimizing total link cost or the maximum link utilization.*

Proof: Given any integer link weighting and the corresponding flow for OSPF, assuming integer $w_{u,v}$ is chosen as the weight for link (u, v), we can assign weight $a \cdot w_{u,v}$ to link (u, v) for DEFT whereas a is a constant number. Since $w_{u,v}$ is integer, the gap of shortest distance of a link along a non-shortest path is at least 1 for OSPF and such gap is at least a for DEFT. Thus the flow proportion of a link along a non-shortest path will be less than e^{-a} of the flow proportion of the link along the shortest path from (4). When a is large enough, e^{-a} is very close to 0, e.g., $e^{-16} \approx 10^{-7}$. Therefore, the flow along any non-shortest path is negligible and DEFT has almost the same flow as OSPF. i.e., DEFT degenerates into OSPF. Therefore, DEFT is no worse than OSPF. ■

In addition, from Theorem 2, DEFT can realize optimal routing for some scenarios where OSPF cannot.

***Theorem** 4: For any traffic matrix, DEFT can determine a unique flow for a given link weighting within polynomial time.*

Proof: Given any link weighting $\mathbf{W}$, the splitting of the flow destined to node t is independent of that of other destinations. For a particular destination and link weighting $\mathbf{W}$, we can determine and split the incoming flow of each node in the decreasing order of its shortest distance to (i.e., starting from the farthest node). This procedure completes within polynomial time. ■

III. DEFT: Optimization Formulation and Solution

In this section, we address how to determine link weights for an arbitrary network topology and traffic matrix, i.e., the scenario with multiple destinations. It is also the most challenging part of all link-weight-based traffic engineering schemes. Previous schemes (e.g., [3], [6], [7]) start from a set of link weights which determine the flow of traffic, and then tune the weights of some links to diversify the traffic. In this work, we develop an optimization formulation where both link weighting and traffic flows are variables at the same time, coupled through constraints in the formulation. Therefore, the solution to the formulation will bring the optimal link weights at once. The resulting optimization problem could be solved as fast as OSPF local search and leads to a much lower congestion cost value. We will present the optimization formulation under DEFT and propose a two-stage iterative method to solve the problem.

A. Novel Optimization Formulation

First, note that it is still difficult to directly integrate the exponentially-weighted flow splitting (4) of DEFT into an optimization formulation because of its discrete feature, i.e. the traffic destined to node t can be sent through link (u, v) if and only if $d_u^t > d_v^t$. Instead of introducing some binary variables, we relax (4) into (6) first, and then by properly setting the lower bound of all link weights, a constant parameter w_{min}, make such a relaxation as tight as we want.

$$\Gamma(h_{u,v}^t) = e^{-h_{u,v}^t} \tag{6}$$

For example, in a flow solution satisfying (6), if there is a link (u, v) where $d_v^t \geq d_u^t$ and $f_{u,v}^t > 0$, then $f_{u,v}^t \leq f_u^t\, e^{-h_{u,v}^t} = f_u^t\, e^{-(d_v^t + w_{u,v} - d_u^t)} \leq f_u^t\, e^{-w_{min}}$. If w_{min} is large enough, this flow portion, which is infeasible to DEFT on link (u, v), could be neglected.

Therefore, we present the following optimization problem **ORIG** (7) using the relaxed rule of flow splitting (i.e., (6)) as the approximation for the traffic engineering under DEFT.

$$\begin{aligned}
\textbf{minimize} \quad & \sum_{(u,v)\in\mathbb{E}} \Phi(f_{u,v}, c_{u,v}) && \text{(7a)}\\
\textbf{subject to} \quad & \sum_{z:(y,z)\in\mathbb{E}} f_{y,z}^t - \sum_{x:(x,y)\in\mathbb{E}} f_{x,y}^t = D(y,t), \forall y \neq t && \text{(7b)}\\
& f_{u,v} = \textstyle\sum_{t\in\mathbb{V}} f_{u,v}^t, && \text{(7c)}\\
& h_{u,v}^t = d_v^t + w_{u,v} - d_u^t, && \text{(7d)}\\
& f_{u,v}^t = f_u^t\, e^{-h_{u,v}^t}, && \text{(7e)}\\
& f_u^t = \max_{(u,v)\in\mathbb{E}} f_{u,v}^t, && \text{(7f)}\\
\textbf{variables} \quad & w_{u,v} \geq w_{min}, f_u^t, d_u^t, h_{u,v}^t, f_{u,v}^t, f_{u,v} \geq 0 && \text{(7g)}
\end{aligned}$$

Constraint (7b) is to ensure flow conservation at an intermediate node y. Constraint (7c) is for flow aggregation on each link. Constraint (7d) is from the definition of the gap of shortest distance. Constraints (7e)-(7f) come from (3) and (6). In addition, (7e) and (7f) also imply that $f_{u,v}^t \leq f_u^t$ and $h_{u,v}^t$ should be 0 for at least one outgoing link (u, v) of node u destined to node t, i.e., the link (u, v) is on the shortest path from node u to node t.

B. Two-Stage Iterative Method

Problem ORIG (7) is non-smooth and non-convex due to the non-smooth constraint (7f) and the nonlinear equality (7e). No tractable general-purpose solver can be applied to this problem directly. We propose a new two-stage iterative method to solve problem ORIG.

First, we relax constraint (7f) into (8) below

$$f_u^t \leq \sum\nolimits_{(u,v)\in\mathbb{E}} f_{u,v}^t, \ \forall t \in \mathbb{V}, \ \forall u \in \mathbb{V}. \tag{8}$$

Eqs. (7a)-(7e), (7g) and (8) constitute problem **APPROX**.

Note that we only need to obtain a "reasonably" accurate solution (link weighting $\mathbf{W}$) to problem APPROX since the inaccuracy caused by the relaxation (8) will be compensated by the successive refinement process. From the $\mathbf{W}$, we can derive the shortest path tree $\mathbb{T}(\mathbf{W}, t)$[4] for each destination t, and all other dependent variables $(d_u^t, h_{u,v}^t, f_u^t, f_{u,v}^t, f_{u,v})$ within DEFT according to Theorem 4. We then use these values as the initial point (which is also strictly feasible) for a new problem **REFINE**, which consists of Eqs. (7a)-(7e), (7g) and (9) below:

$$f_u^t = f_{u,v}^t, \forall t \in \mathbb{V} \cap \forall u \in \mathbb{V} \cap (u, v) \in \mathbb{T}(\mathbf{W}, t). \tag{9}$$

With the two-stage iterative method, we are left with two optimization problems, APPROX and REFINE, both of which

[4] To keep $\mathbb{T}(\mathbf{W}, t)$ as a tree, only one downstream node is chosen if a node can reach the destination through several downstream nodes with the same distance.

have convex objective functions and twice continuously differentiable constraints. To solve the large-scale non-linear problems APPROX and REFINE (with $O(|V||E|)$ variables and constraints), we extend the primal-dual interior point filter line search algorithm, IPOPT [13], by solving a set of barrier problems for a decreasing sequence of barrier parameters μ converging to 0. (See more discussion in Appendix B.)

In summary, in solving problem APPROX, we mainly want to determine the shortest path tree for each destination (i.e., deciding which outgoing link should be chosen on the shortest path). Then in solving problem REFINE, we can tune the link weights (and the corresponding flow) with the same shortest path trees as in APPROX.

Note that the line search approach adopted to solve both APPROX and REFINE could update all link weights simultaneously within one iteration using the general descent method. In contrast, for the local-search techniques [2], each iteration of the search evaluates a candidate solution (i.e., an assignment of the link weights) and sets the stage for exploring a neighborhood of solutions by changing one, or a few, link weights. Therefore, our approach requires fewer iterations than the local-search techniques in general.

C. Pseudocode for Two-Stage DEFT

The pseudocode of the proposed two-stage iterative method for DEFT is shown in Algorithm 1 and 2. Most instructions are self-explanatory. Function DEFT_FLOW(**W**) is described in Theorem 4 to derive a flow from a set of link weights, **W**. Given the initial and ending values for barrier parameter μ, maximum iteration number, with/without initial link weighting/flow, function DEFT_IPOPT() returns a new set of link weights as well as a new flow. Note that, as shown in Algorithm 2, when DEFT_IPOPT() is used for problem APPROX, it returns with the last iteration rather than the iteration with the best $\mathbf{Flow}_i$ in terms of the objective value as in problem REFINE. This is because problem APPROX has different constraints from problem ORIG and an over-aggressive method may leave small search freedom for the successive REFINE problem. Finally, to execute function Two_Stage() as in Algorithm 1, we need to specify initial and terminative μ values, ($\mu_{\text{init}} \geq \mu_{\text{end\_approx}} \geq \mu_{\text{end\_refine}}$), and maximum iteration number $\text{Iter}_{\text{approx}} \geq \text{Iter}_{\text{refine}}$. As to be shown in the later performance evaluation, it is straightforward to specify these parameters.

Algorithm 1 Two_Stage($\mu_{\text{init}}, \mu_{\text{end\_approx}}, \mu_{\text{end\_refine}}, \text{Iter}_{\text{approx}}, \text{Iter}_{\text{refine}}$)

```
$(\mu, \mathbf{W}) \leftarrow$ DEFT_IPOPT($\mu_{\text{init}}, \mu_{\text{end\_approx}}, \text{Iter}_{\text{approx}}$, **nil**)
Initial_Point $\leftarrow (\mathbf{W}, \text{DEFT\_FLOW}(\mathbf{W}))$
$(\mu, \mathbf{W}) \leftarrow$
    DEFT_IPOPT($\mu, \mu_{\text{end\_refine}}, \text{Iter}_{\text{refine}}$, Initial_Point)
return $(\mathbf{W}, \text{DEFT\_FLOW}(\mathbf{W}))$
```

IV. Performance Evaluation

Algorithm 2 DEFT_IPOPT ($\mu_{\text{start}}, \mu_{\text{end}}, \text{Iter}_{\text{max}}$, Initial_Point)

```
**if** Initial_Point $\neq$ **nil then**
    Initiate the problem with Initial_Point /*REFINE*/
**end if**
**for each** iteration $i \leq \text{Iter}_{\text{max}}$ with $\mu_{\text{start}} \geq \mu \geq \mu_{\text{end}}$ **do**
    $\mu_i \leftarrow$ current value for $\mu$
    $\mathbf{W}_i \leftarrow$ current values for all $w_{u,v}$
    $\mathbf{Flow}_i \leftarrow \text{DEFT\_FLOW}(\mathbf{W}_i)$
**end for**
**if** Initial_Point $=$ **nil then**
    return $(\mu_i, \mathbf{W}_i)$ of the last iteration /*APPROX*/
**else**
    return $(\mu_i, \mathbf{W}_i)$ of the iteration with the best $\mathbf{Flow}_i$ in
    terms of objective value /*REFINE*/
**end if**
```

In this section, we present the numerical results of various schemes under many practical scenarios. We employ the same cost function (1) as in [3]. The primary metric used is the optimality gap, in terms of total link cost, compared against the value achieved by optimal routing (determined by the centralized solution to the linear program in Appendix A using CPLEX 9.1 [14] via AMPL [15]). The secondary metric used is the maximum link utilization. We do not reproduce the performance of some obvious link-weight-based traffic engineering approaches for OSPF, e.g., UnitOSPF (setting all link weights to 1), RandomOSPF (choosing the weights randomly), InvCapOSPF (setting the weight of an link inversely proportional to its capacity as recommended by Cisco), L2OSPF (setting the weight proportional to its physical Euclidean distance) [3], since none of them performs as well as the state-of-the-art local-search method in [3]. In addition, since DEFT is never worse than OSPF in terms of minimizing the maximum link utilization or the sum of link cost (Theorem 3), we bypass the scenarios where OSPF can achieve near optimal solution. Instead, we are particularly interested in those scenarios that OSPF does not perform well.

For fair comparisons, we use the same topology and traffic matrix as those in [3]. The 2-level hierarchical networks were generated using GT-ITM, which consists of two kinds of links: local access links with 200-unit capacity and long distance link with 1000-unit capacity. And in the random topologies, the probability of having an link between two nodes is a constant parameter and all link capacities are 1000 units.

Although AT&T's proprietary code for local search used in [3] is not publicly available, there is an open-source software project with IGP weight optimization, TOTEM 1.1 [16]. It follows the same approach as [3], and has similar quality of the results. It is slightly slower due to the lack of the implementation of dynamic Dijkstra algorithm. We use the same parameter setting for local search as in [2], [3] where link weight is restricted as an integer from 1 to 20, initial link weights are chosen randomly, and the best result is collected after 5000 iterations.

To implement the proposed two-stage iterative method for DEFT as shown in Algorithms 1 and 2, we modify another open source software, IPOPT 3.1 [17], and adjust its AMPL interface to integrate it into our test environment. We choose $\mu_{\text{init}} = 0.1$ for most cases except for $\mu_{\text{init}} = 10$ for the 100-node network with heavy traffic load (the last three points of DEFT shown in Fig. 8). We also choose $\mu_{\text{end\_approx}} = 10^{-4}, \mu_{\text{end\_refine}} = 10^{-9}$, and maximum iteration number $\text{Iter}_{\text{approx}} = 1000, \text{Iter}_{\text{refine}} = 400$. The code terminates earlier if the optimality gap has been less than 0.1%.

A. Optimality Gap and Max Link Utilization in Minimizing Total Link Cost

The results for a 2-level topology with 50 nodes and 212 links with seven different traffic matrices are shown in Table II. The results are also depicted graphically in Fig. 4. Besides the two metrics (maximum link utilization and optimality gap in terms of total link cost), we also show the average link utilization under optimal routing as an indication of network load. From the results, we can observe that the gap between OSPF and optimal routing can be very significant (up to 222.8%) for a practical network scenario, even when the average link utilization is not very high ($\leq 27\%$). In contrast, DEFT can achieve almost the same performance as the optimal routing in terms of both total link cost and maximum link utilization.

TABLE II

Results of 2-level topology with 50 nodes and 212 links

Total Demand	1700	2000	2200	2500	2800	3100	3400
Ave Link Load-OPT	0.128	0.148	0.17	0.192	0.216	0.242	0.267
Max Link Load-OPT	0.667	0.667	0.667	0.9	0.9	0.9	0.9
Opt. Gap-OSPF	2.8%	4.4%	7.2%	9.4%	20.7%	64.2%	222.8%
Opt. Gap-DEFT	0.1%	0.1%	0.1%	0.1%	0.1%	0.1%	0.1%

Similar observation hold for other scenarios as shown in Fig. 4-8. Without exception, the curves of the DEFT scheme (the horizontal lines coinciding with x-axes) almost completely match those of optimal routing measured by total link cost and maximum link utilization. Note that, within those figures, the maximum optimality gap of OSPF is as high as 252% in Fig. 7 and that of DEFT is only up to 1.5% in Fig. 8. In addition, DEFT reduces the maximum link utilization compared to OSPF on all tests, and substantially on some tests. However, maximum link utilization is not a metric as accurate as total link cost. For example, in Fig. 4, when the traffic demands uniformly increase by 1/3, the maximum link utilization under optimal routing remains at 90%.

B. Convergence Behavior

Fig. 9 shows the optimality gap achieved by local search OSPF and DEFT, as well as the value of the barrier parameter μ within the first 500 iterations for a typical scenario (corresponding to the points in Fig. 4 with the largest traffic demand). For OSPF local search, the optimality gap is still 386% after 500 iterations, and it takes another 4500 iterations to reduce the optimality gap to 223% (as shown at Fig. 4). In contrast, DEFT can reduce the gap to 13.1% at the end of the APPROX procedure (after 359 iterations). Resuming the searching with $\mu = 10^{-4}$, the REFINE procedure further reduces the gap to 0.1% with only additional 108 iterations.

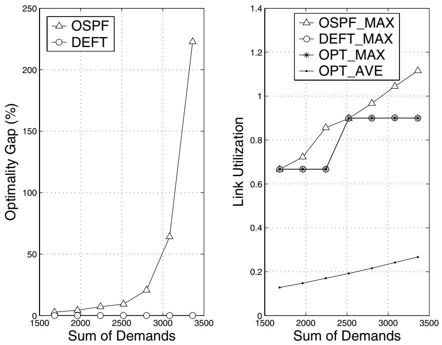

Fig. 4. Comparison of DEFT and Local Search OSPF in terms of optimality gap and maximum link utilization for a 2-level topology with 50 nodes and 212 links

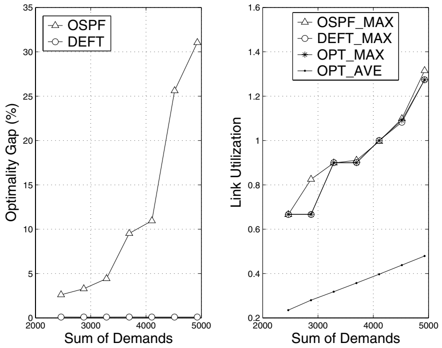

Fig. 5. 2-level topology with 50 nodes and 148 links

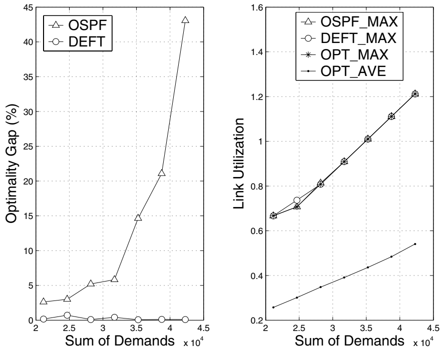

Fig. 6. Random topology with 50 nodes and 228 links

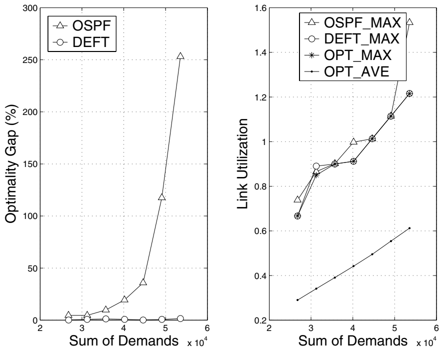

Fig. 7. Random topology with 50 nodes and 245 links

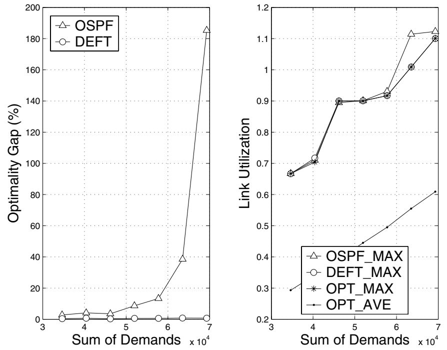

Fig. 8. Random topology with 100 nodes and 403 links

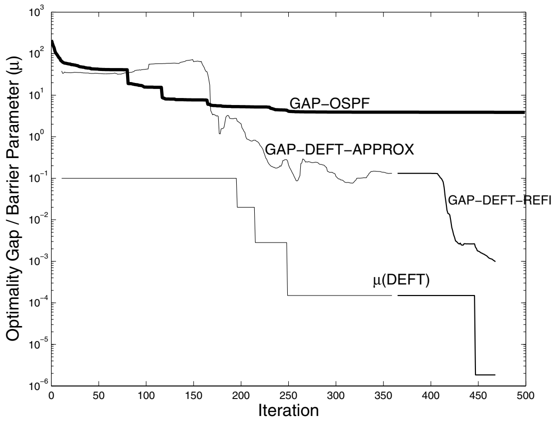

Fig. 9. Evolution of barrier parameter μ in DEFT and comparison of the drop in optimality gap between Local Search OSPF and Two-Stage DEFT in a 2-level topology with 50 nodes and 212 links

Therefore, DEFT converges much faster than local-search method and exhibits an important feature desirable in all optimization algorithms: the ability to provide multiplicative reduction in optimality gap while approaching toward the optimum. This is in part because we incorporate the relationship between link weighting and the flow of traffic into the optimization formulation itself from the beginning.

C. Running Space and Time Requirement

The tests for DEFT and local search OSPF were performed under the time-sharing servers of Redhat Enterprise Linux 4 with Intel Pentium IV processors at 2.8∼3.2 Ghz. The local-search code for OSPF is integrated with TOTEM, which consumes about 700MB memory for all the tested scenarios, and the memory occupied by DEFT varies from 175MB to 2077MB depending on the network size. Note that both local-search code [16] used in OSPF and IPOPT code used in DEFT available to us can be further optimized for speed. Moreover, the running time is also sensitive to traffic matrix since a solution with acceptable optimality can be reached very fast for light traffic matrices. Therefore, we just show their average running time per iteration for qualitative reference.

Table III shows the running time for different networks. We observe that the running time per iteration of DEFT is comparable with local search OSPF but the iteration number required for DEFT (at most 1400 iterations and as low as 271 iterations in our tests) is much less than that for local search OSPF (5000 iterations). Therefore, DEFT is very promising to achieve near optimal traffic engineering within a reasonable time, even for large networks.

TABLE III

Average running time per iteration and number of iterations required by DEFT and local search OSPF to attain the performance in Fig. 4-8

Net. Type	Node	Link	Time per Iteration (s)		Iteration	
			DEFT	OSPF	DEFT	OSPF
2-level	50	148	0.7∼3.5	6.0∼13.9	271∼825	5000
2-level	50	212	1.0∼4.8	6.4∼17.4	308∼1020	5000
Random	50	228	3.3∼5.0	3.2∼9.0	400∼1400	5000
Random	50	245	6.0∼12.3	6.1∼14.1	620∼1400	5000
Random	100	403	59∼126	39.5∼105.1	479∼994	5000

V. Conclusion and Future Work

Network operators today try to alleviate congestion in their own network by tuning the intra-domain routing parameters. Unfortunately, traffic engineering under OSPF or IS-IS to avoid network-wide congestion is computationally intractable, forcing the use of local-search techniques. We propose a new routing scheme called DEFT: Distributed Exponentially-weighted Flow SpliTting. The success of DEFT can be attributed to two additional features. First, in terms of protocol adjustment, DEFT can put traffic on non-shortest paths, with an exponential penalty on longer paths. Second, in terms of computational method, DEFT solves the resulting optimization problem by integrating link weights and the corresponding traffic distribution together in the formulation. This formulation leads to a much more efficient way of tuning link weights than the existing local-search heuristic for OSPF. Collectively, these features enable DEFT to substantially reduce optimality gap to near-zero with a running time similar to or faster than OSPF local search.

DEFT is readily implementable as an extension to existing IGPs. It is provably better than OSPF in minimizing the sum of link cost. DEFT retains the simplicity of having routers compute paths based on configurable link weights, while approaching the performance of more complex routing protocols that can split traffic arbitrarily over any paths.

In this paper, we only address the link weighting under DEFT for a given traffic matrix. The next challenge would be to explore robust optimization under DEFT, optimizing to select a single weight setting that works for a range of traffic matrices and/or a range of link/node failure scenarios.

In this case of “Design for Optimizability”, a simple change to link-state-based routing protocol leads to a much more solvable optimization problem, which, together with the computational method of two-stage relaxation, leads to DEFT.

Acknowledgment

This research is in part supported by the National Science Foundation (NSF) under the contracts CNS-0519880 and CCF-0635034, and Cisco URP grant GH072605. We would like to thank Bernard Fortz and Hakan Umit for providing the code for local search OSPF and the network topology, which are used in our simulation study for a fair comparison between DEFT and OSPF. We also appreciate the helpful discussions on large-scale non-linear optimization with Sven Leyffer, Andreas Wäechter, Gabriel Lopez Calva, Richard Waltz, and Robert J. Vanderbei. Finally, we acknowledge the valuable comments from Jiayue He.

Appendix

A. Integer Linear Program for OSPF and Linear Program for Optimal Routing

$$\min \quad \sum_{(u,v)\in\mathbb{E}} \Phi(f_{u,v}, c_{u,v}) \tag{10a}$$

$$\text{s.t.} \quad \sum_{z:(y,z)\in\mathbb{E}} f^t_{y,z} - \sum_{x:(x,y)\in\mathbb{E}} f^t_{x,y} = D(y,t), \forall y \neq t \tag{10b}$$

$$f_{u,v} = \sum_{t\in\mathbb{V}} f^t_{u,v}, \tag{10c}$$

$$h^t_{u,v} = d^t_v + w_{u,v} - d^t_u, \tag{10d}$$

$$f^t_{u,v} \leq f^t_u, \tag{10e}$$

$$h^t_{u,v} \leq M(1 - \delta^t_{u,v}), \tag{10f}$$

$$f^t_u - f^t_{u,v} \leq M(1 - \delta^t_{u,v}), \tag{10g}$$

$$1 - \delta^t_{u,v} \leq M h^t_{u,v}, \tag{10h}$$

$$f^t_{u,v} \leq M\delta^t_{u,v}, \tag{10i}$$

$$\text{vars.} \quad w_{u,v}, f^t_u, d^t_u, h^t_{u,v}, f^t_{u,v}, f_{u,v} \geq 0, \tag{10j}$$

$$\delta^t_{u,v} \in \{0, 1\}. \tag{10k}$$

The integer linear program formulation to search for the best link weights under OSPF is shown at (10). Eqs. (10a)-(10d) are copied from (7). M is a very large constant positive number to deal with binary variables and $\delta^t_{u,v}$ is a binary variable to represent if link (u,v) is on the shortest path from u to t. Thus, if $\delta^t_{u,v} = 1$, then $h^t_{u,v} = 0$ due to (10f) and $f^t_{u,v} = f^t_u$ due to (10e) and (10g) while if $h^t_{u,v} = 0$, then $\delta^t_{u,v} = 1$ due to (10h). On the contrary, if $\delta^t_{u,v} = 0$ then $f^t_{u,v} = 0$ due to (10i). Therefore, formulation (10) realizes the equal flow splitting across multiple shortest paths under OSPF. Note that, we do not limit the link weights $w_{u,v}$ to integer values to speed up the searching procedure. The resulting non-integer path lengths could be treated as equal if they differ by less than a specified tolerance as in [8].

In addition, the linear program for optimal routing consists of (10a)-(10c).

B. IPOPT: Primal-dual Interior Point Filter Line Search

The two optimization problems, APPROX and REFINE, discussed in Sec. III can be transformed into a general formulation (11) below.

$$\min \quad f(\mathbf{x}) \tag{11a}$$

$$\text{s.t.} \quad c(\mathbf{x}) = 0 \tag{11b}$$

$$\text{vars.} \quad \mathbf{x} \succeq 0 \tag{11c}$$

where both $f(\mathbf{x})$ and $c(\mathbf{x})$ should be twice continuously differentiable. The method in [13] calculates solutions for a set of barrier problems (12) for a decreasing sequence (with a superlinear rate) of barrier parameters μ converging to 0.

$$\min \quad \varphi_\mu(\mathbf{x}) \triangleq f(\mathbf{x}) - \mu \sum_i \ln(x_i) \tag{12a}$$

$$\text{s.t.} \quad c(\mathbf{x}) = 0 \tag{12b}$$

The primal-dual equations are shown at (13) below

$$\nabla f(\mathbf{x}) + \nabla c(\mathbf{x})\lambda - \mathbf{z} = 0 \tag{13a}$$

$$c(\mathbf{x}) = 0 \tag{13b}$$

$$\mathbf{diag}(\mathrm{x})\mathbf{diag}(\mathbf{z})\mathbf{e} - \mu\mathbf{e} = 0 \tag{13c}$$

where $\mathbf{e}$ is the vector of all ones, λ and $\mathbf{z}$ are the Lagrangian multipliers for the equality constraints (11b) and the bound constraints (11c). The method in [13] computes an approximation solution to the barrier problem (12) for a barrier parameter μ using a damped Newton's method, and uses the solution as the initial point for the next barrier problem with a smaller μ value. Further description can be found in [13].

References

[1] D. Awduche, A. Chiu, A. Elwalid, I. Widjaja, and X. Xiao, "Overview and Principles of Internet Traffic Engineering," IETF, RFC 3272, May 2002.

[2] B. Fortz and M. Thorup, "Internet traffic engineering by optimizing OSPF weights," in *INFOCOM'00, Tel Aviv, Israel*, 2000, pp. 519–528.

[3] ——, "Increasing Internet capacity using local search," *Computational Optimization and Applications*, vol. 29, no. 1, pp. 13–48, 2004.

[4] A. Feldmann, A. G. Greenberg, C. Lund, N. Reingold, J. Rexford, and F. True, "Deriving traffic demands for operational IP networks: methodology and experience." *IEEE/ACM Transactions on Networking*, vol. 9, no. 3, pp. 265–280, 2001.

[5] B. Fortz, J. Rexford, and M. Thorup, "Traffic engineering with traditional IP routing protocols," *IEEE Communication Magazine*, vol. 40, no. 10, pp. 118–124, Oct. 2002.

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

[7] L. Buriol, M. Resende, C. Ribeiro, and M. Thorup, "A memetic algorithm for OSPF routing," in *Proceedings of the 6th INFORMS Telecom*, 2002, pp. 187–188.

[8] A. Sridharan, R. Guérin, and C. Diot, "Achieving near-optimal traffic engineering solutions for current OSPF/IS-IS networks," *IEEE/ACM Transactions on Networking*, vol. 13, no. 2, pp. 234–247, 2005.

[9] D. Awduche, "MPLS and traffic engineering in IP networks," *IEEE Communication Magazine*, vol. 37, no. 12, pp. 42–47, Dec. 1999.

[10] Z. Wang, Y. Wang, and L. Zhang, "Internet traffic engineering without full mesh overlaying," in *INFOCOM'01, Anchorage, Alaska*, Apr. 2001.

[11] J. H. Fong, A. C. Gilbert, S. Kannan, and M. J. Strauss, "Better alternatives to OSPF routing," *Algorithmica*, vol. 43, no. 1-2, pp. 113–131, 2005.

[12] Z. Cao, Z. Wang, and E. W. Zegura, "Performance of hashing-based schemes for Internet load balancing," in *INFOCOM'00, Tel Aviv, Israel*, 2000, pp. 332–341.

[13] A. Wächter and L. T. Biegler, "On the implementation of a primal-dual interior point filter line search algorithm for large-scale nonlinear programming," *Mathematical Programming, 106(1)*, pp. 25–57, 2006.

[14] ILOG CPLEX, http://www.ilog.com/products/cplex/.

[15] R. Fourer, D. M. Gay, and B. W. Kernighan, *AMPL: A Modeling Language for Mathematical Programming*. Danvers, MA, USA: Boyd & Fraser Publishing Co., 1993.

[16] TOTEM, http://totem.info.ucl.ac.be.

[17] IPOPT, http://projects.coin-or.org/Ipopt.