

Revealing the Optimality Gap for Traffic Engineering Algorithms

Huan Liu
Accenture Technology Labs
huan.liu@accenture.com

Abstract—Traffic Engineering (TE) is important and necessary in order to fully utilize the existing network resources and reduce capital expenditure. Given its importance, many algorithms have been proposed in the literature. Unfortunately, there is still not a consistent methodology to evaluate these algorithms. Worse yet, some variants of the traffic engineering problem are known to be NP-hard. Thus, given a particular TE problem, in general, it is not possible to know the optimal solution; hence, it is difficult to assess how a particular heuristic algorithm performs. Even though several heuristic algorithms could be used to validate each other, the best solution from these algorithms could still be very far away from the optimal solution. We propose a novel methodology to evaluate TE algorithms. In this methodology, we construct TE problems with known optimal solutions and we then use these TE problem instances to test the performance of TE algorithms. We found that some TE algorithms perform poorly, and the result deviates from the optimum further as the problem size gets bigger. Our results suggest that there is large room for algorithm improvements and further research is required. Even though we only demonstrate the power of the methodology in the context of traffic engineering algorithms, the methodology is general enough that it could be applied in many other areas as well.

I. INTRODUCTION

Internet Service Providers (ISP) see a dramatical growth in Internet traffic in the backbone over the past few years. This increase can be attributed to many different reasons. First, video sharing over the Internet (e.g., YouTube) becomes increasingly popular. Because video files are typically very big, their transmission would consume a significant amount of bandwidth. Second, broadband is continuing its fast pace of adoption. As reported by Nielsen/NetRatings, US broadband penetration broke 80% among active Internet users in February 2007. Third, the per person usage of Internet keeps on increasing. We increasingly rely on the Internet, from communication (email), learning, entertainment to online shopping.

The increase in traffic poses a big challenge to ISPs. Provisioning more capacities is certainly one possible solution, however, it is not very economical for two reasons. First, a lot of capacities have to be provisioned due to the rapid rise in traffic. Second, the revenue per bit of traffic continues to decrease rapidly, making it harder to recoup the cost of the capital investment.

Instead of making costly capital investments, an alternative solution is to make more efficient use of the existing network by using Traffic Engineering (TE) techniques. Traffic Engineering (TE) determines how to route traffic in a network in order to achieve certain optimization objectives, such as

minimizing the maximum link utilization or minimizing the total delay. It is an important tool for the ISPs in order to fully utilize the existing network capacity and reduce the need for future capital investment.

Without traffic engineering, Internet traffic is routed based on shortest paths, which could lead to inefficient use of the provisioned capacity. ISPs typically employ OSPF (or IS-IS) routing protocol in their backbone network. Under the OSPF protocol, the routers exchange neighboring link information, and at the end, each router knows about all the links in the network and their associated metrics. When routing, each router will send the packet along the shortest path based on the link metrics to the destination node. Since traffic is routed oblivious of the link condition (always shortest path regardless of whether a link is congested or not), some links could be overloaded, while other links are under-utilized.

There are two ways to perform traffic engineering: IGPWO TE or MPLS TE. IGPWO (Interior Gateway Protocol Weight Optimization) TE computes the link metrics such that traffic is evenly distributed when routed using the shortest path [1] [2]. MPLS TE employs MPLS technology, by which packets are routed along explicit paths based on a predefined set of labels. These explicit paths are referred to as the Label Switched Path (LSP). Because the paths are explicitly specified, routing is no longer constrained to be the shortest path; hence, more efficient use of the network becomes possible.

Although many TE algorithms have been proposed in the literature, including algorithms for IGPWO TE [1] [2] and algorithms for MPLS TE [3] [4] [5] [6] [7], there have not been any comprehensive comparisons. One can certainly compare all the existing algorithms against each other. However, for this approach to be valuable, we have to agree upon a standard set of representative test cases, which is very hard to achieve due to the diversity of ISP networks and the rapid changing traffic demands. Even if we are able to compare the algorithms with each other, we still do not know how well they perform with regard to the optimum. They could all produce similar results, however, the results could still be very far away from the optimum.

In this paper, we propose a novel methodology to test the effectiveness of TE algorithms. Our methodology first constructs test instances with a known optimal solution, then these test instances are solved by the TE algorithm. By comparing the result from the algorithm and the optimal solution, we can reveal the optimality gap, which is the difference between

the optimum and the solution from a TE algorithm. The optimality gap is a good indicator of the effectiveness of the algorithm. A consistent small gap means the algorithm performs well, whereas, a large gap indicates that there is room for improvement.

In addition to being a valuable testing tools, the methodology also serves as a debugging tool. For heuristic algorithms, if they produce results that are better than the optimum, we know there is a bug in the implementation. Similarly, for exact algorithms, we know something is wrong if they produce results other than the optimum. In fact, this methodology has been applied to algorithms developed internally in a large ISP and it helped to uncover several real bugs.

Although smaller manually constructed test cases with known optimum sometimes are used during TE algorithm development, they are quite different from our methodology. First, our methodology can generate arbitrarily large test cases, which enables us to study how the TE algorithms scale. Second, our methodology can automatically generate a large number of instances, which enables us to study the statistical trend.

Since our methodology can only generate test cases with a known optimal result, it certainly restrict the set of test cases that could be used. However, we believe our methodology is still widely applicable because of our construction method. The user could supply a real topology and a real set of traffic demands, and the construction method will only change the link capacity or add some fake traffic demands depending on the optimization objective.

This paper is organized as follows. In Sec. II, we first describe our proposed testing methodology. The methodology is based on the assumption that we can construct test scenarios with a known optimal solution. In Sec. III, we describe how to construct MPLS TE test problems. With a simple extension, our methodology can be used to construct IGPWO TE test problems as well. However, due to space limitation, we omit the details. Using our proposed testing methodology, we evaluate several TE algorithms. We report our findings in Sec. IV. Lastly, we conclude in Sec. V.

II. THE PROPOSED TESTING METHODOLOGY

The most commonly used methodology to evaluate an algorithm is depicted in Fig. 1. In step 1, a user first come up with an arbitrary TE problem to serve as a test. In step 2, the test problem is solved optimally, possibly through exhaustive enumeration. As we will show, some variants of the TE problem is NP-complete; hence, this step could take a long time. In step 3, the TE algorithm under test is used to solve the same test problem. Lastly, in step 4, the results are compared to evaluate the effectiveness of the TE algorithm. A major drawback of this methodology is that the optimal solution is often not known, and deriving the optimal solution would take a long time, or it is just plainly not computationally feasible.

Motivated by the shortcomings of the existing methodology, we propose the following new methodology as depicted in Fig. 2. In step 1, instead of starting from an arbitrary TE

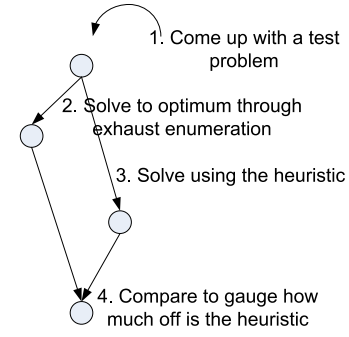


Fig. 1. The most commonly used testing methodology.

problem, we start from an optimal solution. Then in step 2, we construct a TE problem instance from the optimal solution. Step 3 and 4 are the same as before, except that we now know the exact optimal solution by construction.

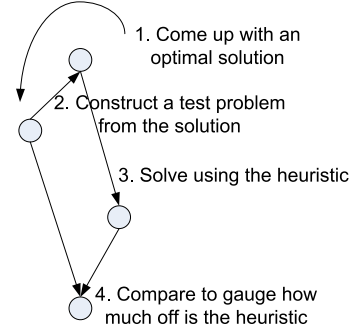


Fig. 2. The proposed testing methodology.

The difficulty in implementing the new methodology lies in constructing examples with known optimal solutions. In the following two sections, we describe how we construct these test problems for MPLS TE and IGPWO TE respectively.

III. CONSTRUCTING TEST MPLS TE PROBLEMS

A. Definition of MPLS TE

Before we show how to construct test problems, we first precisely define what is the MPLS TE problem. The most basic version of the MPLS TE problem can be stated as an optimization problem as follows.

- **Input:**

- 1) The network topology along with the capacity for each link. The network has N nodes and M link.
- 2) The set of traffic demands that need to be supported in the network. These traffic demands are expressed as a set of LSPs to be established, where each LSP specifies the source and destination nodes and the bandwidth requirement.

- **Output:** The set of traffic demands that can be routed in the network and the route assignment for each of them.

- **Optimization objectives:** Many optimization objectives are possible. We consider four popular objectives:

- *Objective 1*: Maximizing the absolute minimum free bandwidth over all links
- *Objective 2*: Minimizing the maximum percentage utilization over all links
- *Objective 3*: Maximizing the number of traffic demands that can be supported in the network
- *Objective 4*: Minimizing the total delay.

We will postpone the exact mathematical definition for these objectives until when we describe how we construct test problems.

We distinguish between two cases of the MPLS TE problem, which differ in how the traffic demands are given. In the first case, all traffic demands are given at the beginning, and hence, the MPLS TE algorithm can optimize the routing taking into account all traffic demands. We refer to this case as *static MPLS TE*. In the second case, the traffic demands are given one by one to the MPLS TE algorithm, and the MPLS TE algorithm has to determine how to route the current traffic demand without disturbing existing traffic demands. We refer to this case as *dynamic MPLS TE*.

B. How to construct test problems with known optimums

Let us first understand why the MPLS TE problem is hard to solve, which will motivate our construction method. We will focus on the static MPLS TE problem.

There are two variants of the MPLS TE problem, one allows traffic bifurcation, the other does not. If bifurcation is allowed, a traffic demand could be split across multiple parallel paths, which may be undesirable because of the possibility of packet re-sequencing.

If bifurcation is allowed (i.e., Equal Cost MultiPath (ECMP)), the static MPLS TE problem is simply a generic multi-commodity flow problem, which can be solved in polynomial time. Efficient algorithms, such as flow deviation [8] or Linear Programming (LP), can solve the problem optimally and quickly.

If bifurcation is not allowed, we can easily prove that the static MPLS TE problem becomes NP-hard, meaning that there exists no polynomial time algorithm which can solve the problem optimally. To understand why bifurcation could play such an important role, let us look at a simple example as shown in Fig. 3. There are two nodes in this network and there are two parallel links between these two nodes.

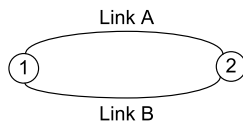


Fig. 3. A simple network.

Let us assume we have four traffic demands between these two nodes, with bandwidth requirements of 5, 6, 7, 8 respectively. Let us also assume that both links have the same capacity of 20. As an example, let us assume we are optimizing for Objective 1: maximize the minimum free bandwidth. How

do we route these traffic demands in order to optimize our objective?

If bifurcation is allowed, the answer is easy because we just route half of the traffic on one link and the other half on the other link. The problem is much harder when bifurcation is not allowed because if we fix a traffic demand on a link, then the whole traffic demand has to be routed on that link, which could lead to uneven link utilization. In order to find the optimum, it is necessary to enumerate all possible ways of routing the demands on the two links, resulting in examining 2^4 combinations. The number of combinations increases exponentially as the number of parallel links and the number of traffic demands increase, making it quickly impractical to enumerate all possibilities.

As we can see, bifurcation turns the problem into a combinatorial problem, meaning that we have to evaluate a large number of combinations in order to find the optimum, since each traffic demand has to be routed as a whole. In fact, to prove that the static MPLS TE problem with bifurcation is NP-complete, we could use the same parallel links construction to turn the TE problem into a packing problem. For each box in the packing problem, we create a parallel link where the link capacity corresponds to the box capacity. Then, for each object, we create a traffic demand whose size corresponds to the object size.

Although it is hard to determine the right combination if we are given a random problem, it is easy to fix a combination if we were to construct the problem. Once we fix the combination, we can adjust the link capacity to make the chosen combination the optimal solution.

Let us look at a concrete example for the network shown in Fig. 3. Let us assume we pick a combination where demands with sizes 5 and 6 are routed in Link A and demands with sizes 7 and 8 are routed in Link B. If we are optimizing for Objective 2: minimize the maximum percentage utilization, we can fix Link A capacity to be $2 \times (5 + 6) = 22$ and Link B capacity to be $2 \times (7 + 8) = 30$, then we know the particular combination we have chosen is the optimum with a maximum link utilization of 50%. After we determine the link capacities, we can feed the constructed test problem to TE tools and test whether they can get to a result that is close to the optimum.

We can of course extend the above example to larger problems. The key is to construct network topologies with many equal-length shortest paths between any given pair of nodes. We can randomly pick one shortest path when routing a traffic demand and size the links appropriately. Since we always choose the shortest path and since the link capacity is chosen to fit the routing solution, we know the particular routing we chose is the optimum.

The construction method can be easily applied to very large problems with a large network topology and many traffic demands. Hence, it will allow us to study the scalability of the TE algorithms.

In the following, we show the construction method for each of the objective functions individually. To be consistent with our notation in Fig. 2, step 1 is on how we construct the

optimal solution and step 2 is on how we construct the test problem from the given solution.

We assume two things are given before we start constructing test problems:

- 1) The network topology, including the number of nodes and the list of links. Any topology could be used to reflect a realistic scenario.
- 2) The set of traffic demands (including the source, destination nodes and the size of the demands) that have to be routed in the network. Again, any set of traffic demands could be used to reflect a realistic scenario.

1) *Objective 1: Maximizing the absolute minimum free bandwidth over all links:* Let l_i be the load (amount of traffic) and c_i be the capacity on link i . Then this objective function can be expressed as:

$$\text{maximize } \min_i c_i - l_i$$

By optimizing for this objective, we make sure that we leave the largest room for any one of the future traffic demands.

Our construction method for Objective 1 is as follows.

- *Step 1:* For each traffic demand, find all possible shortest paths from the source to the destination. Randomly pick one of the shortest paths and route the traffic demand on it.
- *Step 2:* For each link i , sum up the total traffic going through it to determine l_i . Set $c_i = l_i + \delta$, where δ is an arbitrarily chosen constant.

By construction, the chosen parameter δ will be the optimal result. We now prove that it is indeed the optimum.

Theorem 1: For any problem constructed by the method above, the maximum minimum free bandwidth is δ .

Proof: Suppose it is not the case, i.e., there is another solution with $\delta^* > \delta$. The total traffic running in the network in the new solution can be summed up to be:

$$T^* = \sum_i l_i^* \leq \sum_i (c_i - \delta^*)$$

For the solution from the construction method, we can also sum up the total traffic $T = \sum_i (c_i - \delta)$. Since $\delta^* > \delta$, we have $T^* < T$. However, from the construction method, we know that all traffic demands have gone through the shortest path, hence T is the smallest possible, which leads to a contradiction. ■

2) *Objective 2: Minimizing the maximum percentage utilization over all links:* This objective function tries to even out the traffic over all links as a percentage of the link capacity. It can be mathematically expressed as:

$$\text{Minimize } \max_i \frac{l_i}{c_i}$$

If we anticipate future traffic demands to share the same distribution as the given set of traffic demands (or if we assume these traffic demands would scale up proportionally), by optimizing for this objective function, we could accommodate the most amount of future traffic.

Our construction method for this objective is roughly the same as before. The only difference is in step 2, where we set the link capacity differently. The construction steps are as follows:

- *Step 1:* For each traffic demand, find all possible shortest paths from the source to the destination. Randomly pick one of the shortest paths and route the traffic on it.
- *Step 2:* For each link i , sum up the total traffic going through it to determine l_i . Set $c_i = l_i/\theta$, where θ is a constant between 0 and 1.

By construction, the chosen parameter θ will be the optimal result. We now prove that it is indeed the optimum.

Theorem 2: For any problem constructed by the above method, the minimum maximum utilization is θ .

Proof: Suppose it is not the case, i.e., there is another solution with $\theta^* < \theta$. The total traffic running in the network in the new solution can be summed up to be:

$$T^* = \sum_i l_i^* \leq \sum_i c_i \theta^*$$

For the solution from the construction method, we can also sum up the total traffic $T = \sum_i c_i \theta$. Since $\theta^* < \theta$, we have $T^* < T$. However, from the construction method, we know that all traffic demands have gone through the shortest path, hence T is the smallest possible, which leads to a contradiction. ■

3) *Objective 3: Maximizing the number of traffic demands that can be routed:* When the total traffic demands exceed what the network can support, some traffic demands have to be rejected. This objective maximizes the number of satisfied traffic demands; hence, maximizes the number of satisfied customers.

The construction steps are as follows:

- *Step 1a:* Let D be the total number of traffic demands. Pick a constant ρ between 0 and 1, which represents the percentage of traffic demands to satisfy. Sort all D traffic demands based on their shortest path length in the increasing order.
- *Step 1b:* For each one of the first ρD traffic demands, find all possible shortest paths from the source to the destination. Randomly pick one of the shortest paths and route the traffic on it.
- *Step 2:* For each link i , sum up the total traffic going through it to determine l_i . Set $c_i = l_i$.

By construction, ρ would be the optimal solution. We now prove this is the case.

Theorem 3: For any problem constructed by the above method, the maximum number of traffic demands that can be supported is ρD .

Proof: Suppose it is not the case, i.e., there is another solution with $\rho^* > \rho$. Let d_j be the amount of traffic the j th demand generates (sum of bandwidth consumed on each hop) if it is routed on the shortest path. The total traffic running in the network in the new solution can be summed up to be:

$$T^* \geq \sum_{\text{all routed traffic demand } j} d_j \quad (1)$$

$$\geq \sum_{j=1}^{\rho^* D} d_j \quad (2)$$

$$> \sum_{j=1}^{\rho D} d_j = \sum_i l_i = \sum_i c_i$$

The inequality in Equation (1) holds because not all traffic demands are necessarily routed on the shortest path in the new solution. Equation (2) is true because the traffic demands are sorted in increasing order of their length; therefore, any $\rho^* D$ demands will generate at least the same amount of traffic as the first $\rho^* D$ demands.

The equation shows the total traffic generated is more than the network capacity, which leads to a contradiction. ■

4) *Objective 3: Minimizing the total delay:* Packet delay is a very important component of the Service Level Agreement (SLA) the service providers give to their customers. Minimizing the total delay helps to meet these SLA constraints.

The queuing delay through any link is proportional to the percentage load, i.e., $\frac{1}{1-l_i}$. To minimize the network-wide delay, we can minimize the total delay objective function:

$$\text{Minimize } \sum_i \frac{1}{c_i - l_i} \quad (3)$$

We choose this particular form of objective function because the DAMOTE algorithm [9] uses the exact same form. However, our construction method and the following proof on optimality carry over to other forms of the objective function, such as the traffic averaged network-wide delay.

The construction steps are as follows:

- *Step 1:* For each traffic demand, find all possible shortest paths from the source to the destination. Randomly pick one of the shortest paths and route the traffic on it.
- *Step 2a:* For each link i , sum up the total traffic going through it to determine l_i . Determine $l_{\max} = \max_i l_i$.
- *Step 2b:* For each link i , add a traffic demand between the two end nodes with the size of the demand equal to $l_{\max} - l_i$.
- *Step 2c:* The load on all links should now be l_{\max} . Set the capacity on all links to be $c = l_{\max}/\theta$.

All links in the constructed problem should have the same link capacity, which we denote simply by c .

We now show that the constructed problem achieves the optimal result.

Theorem 4: For any problem constructed by the above method, the optimal minimum total delay is $\sum_i \frac{1}{c-c\theta} = \frac{M}{c-c\theta}$.

Proof: Suppose it is not the case, i.e., there is another solution with lower delay. Let l_i^* be the load on link i in this new solution. Then the total delay in the new solution is $\sum_i \frac{1}{c-l_i^*}$.

To compare with the solution in the constructed problem, we need to apply Jensen's inequality, which states that for a convex function $\phi(x)$, the following is true:

$$\frac{\sum \phi(x_i)}{n} \geq \phi\left(\frac{\sum x_i}{n}\right)$$

The equality only holds when all x_i are the same. Let us define $\phi(x) = \frac{1}{1-x}$, which is a convex function. Applying Jensen's inequality, we have

$$\begin{aligned} \sum_i \frac{1}{c-l_i^*} &= \frac{1}{c} \sum_i \frac{1}{1-l_i^*/c} \\ &\geq \frac{M}{c} \frac{1}{\sum_i l_i^*/c} \\ &\quad \frac{1}{1 - \frac{l_{\max}}{c}} \\ &\geq \frac{M}{c} \frac{1}{\sum_i l_i/c} \\ &\quad \frac{1}{1 - \frac{l_{\max}}{c}} \\ &= \frac{M}{c} \frac{1}{1 - l_{\max}/c} = \frac{M}{c - c\theta} \end{aligned} \quad (4)$$

Equation (4) holds true because all traffic demands in the constructed problem are routed along the shortest path; hence, the total network-wide load $\sum_i l_i$ should be the minimum.

The equation above contradicts the assumption. ■

We have shown that the MPLS TE problem with no bifurcation is NP-complete. For these problems, the known optimum is invaluable because we can use it to evaluate the effectiveness of heuristic algorithms. Even in the case of bifurcation, where it is easy to derive the optimum, our methodology is still a valuable debugging tool to check the implementation of algorithms. In fact, our methodology has been applied to several algorithms developed in-house in a large service provider's R&D department, and it has helped to uncover several bugs both in the algorithm with bifurcation and without. These bugs would have been difficult to catch with the traditional methodology or the small number of hand-constructed test cases.

IV. EXPERIMENTAL RESULTS

Using our new methodology, we can evaluate many proposed TE algorithms. We are not only interesting in assessing their performance (i.e., revealing the optimality gap), but we are also interested in seeing how the optimality gap and the computation time scale as the network increases in size.

For our evaluation, we use an open source TE software: TOTEM [9]. TOTEM is a toolbox which integrates many traffic engineering algorithms. It is also extensible such that new algorithms could be integrated easily.

As inputs to our methodology, we need both the network topology and the traffic demands. We generate the topologies using the Waxman model [10]. The Waxman model selects links at random. To make the generated topology resemble real topologies, shorter links are given a higher probability to be chosen. The probabilities are controlled by two parameters: α and β . We choose $\alpha = 0.15$ and $\beta = 0.2$. For all topologies,

we generate twice as many links as the number of nodes, i.e., $M = 2N$. We also generate topologies using BRITTE [11], which creates topologies that resemble the real Internet. Since our observation is largely identical, we omit the results in the interest of space.

We generate one traffic demand for each pair of nodes; hence, there are $N(N-1)/2$ traffic demands. The size of each demand is randomly chosen within the range of [1, 10] (the exact unit does not matter since the link capacity is expressed in the same unit).

For our experimental study, we focus only on dynamic MPLS TE algorithms, but not on static MPLS TE algorithms. The reason is not only because TOTEM has only dynamic MPLS TE implementations, but also because dynamic MPLS TE is the most frequently encountered case in real operational networks. Typically, ISPs route new traffic demands as they come along, and most of the times, they do not want to reroute existing traffic demands. However, it is worth noting that our methodology applies to both the dynamic and static cases.

A. Algorithms evaluated

We evaluate several TE algorithms that are currently implemented in TOTEM. In this section, we give a brief description of each algorithm.

1) *CSPF*: Constrained Shortest Path First (CSPF) algorithm is one of the most used MPLS TE algorithm. In fact, at least one commercial TE tools vendor employees the CSPF algorithm, even for the static MPLS TE case.

As the name implies, CSPF observes constraints, such as the bandwidth constraint, when routing. To make sure the constraints are satisfied, CSPF first prunes all paths between the source and destination that do not satisfy the given constraints. In the case of bandwidth constraint, it will remove all paths that do not have sufficient free capacity. After pruning, CSPF computes the shortest paths based on a defined link metric. If there are many shortest paths, one is randomly chosen.

Different link metrics could be used by CSPF. For our experiments, we use the inverse of the free bandwidth $1/(c_i - l_i)$. We choose this link metric because it corresponds closely to Objective 1: maximizing the minimum free bandwidth. As the free bandwidth of a link shrinks, it becomes increasingly costly to route traffic over it, thus giving strong incentive to route traffic on less loaded links to even out the traffic flow.

2) *DAMOTE*: DAMOTE is similar to CSPF in principle. However, it can perform much clever optimizations based on a network-wide score function. Examples of such functions are: load balancing, hybrid load balancing (where long detours are penalized), pre-emption-aware routing (where rerouting existing LSPs are possible, but penalized). DAMOTE's score function is generic and it is a parameter of the algorithm.

Besides calculating the primary LSPs, DAMOTE can also calculate local or global detour LSPs, which are used to route traffic when some link on the primary LSP fails. However, we will not be using this functionality, as we will not test the ability to calculate backup paths.

We use two score functions for our evaluation. The first is the pure-load-balancing score function, which can be expressed as

$$\sum_i \left(\frac{l_i}{c_i} - \frac{\bar{l}}{c} \right)^2 \quad (5)$$

where $\frac{\bar{l}}{c} = \frac{1}{M} \sum_i \frac{l_i}{c_i}$ is the mean of the relative link load throughout the network. This score function is essentially the variance on the relative link load and, hence, represents the deviation from the optimal load balancing situation. Minimizing this score function is equivalent to Objective 2: Minimizing the maximum percentage utilization.

The second score function is $\sum_i \frac{1}{c_i - l_i}$, which is the same as Objective 4 as shown in equation (3). We use this to test how well DAMOTE minimizes the total delay.

3) *MIRA*: Minimum Interference Routing Algorithms (MIRA) try to route the current traffic demand with the minimum interference (or impact) to future traffic demands [3]. Two MIRA algorithms are included in the TOTEM toolbox: NEWMIRA [4] and Simple MIRA (SMIRA) [5].

Minimizing interference is roughly equivalent to Objective 1: Maximizing the minimum free bandwidth. This is because the more free bandwidth we have, the more likely some future traffic demands can be satisfied.

There are also other algorithms integrated in TOTEM, such as SAMCRA [6] [7]. However, they do not correspond well to any given optimization objective. For example, SAMCRA is concerned with finding a feasible solution that satisfies multiple constraints, but it does not optimize any objective function. Hence, we do not evaluate these algorithms.

B. Scalability

To evaluate the scalability of the algorithms, we generate several test problems with varying network size ranging from $N = 20$ to $N = 500$. We then solve these problems and record the CPU time they take. We solve all problems on an Intel Blade server with a 2.33 GHz Xeon processor.

In Fig. 4, we plot the computation time. The CSPF algorithm is very efficient. We are able to solve a problem with $N = 500$ using roughly 24 hours computation time. In comparison, NEWMIRA and SMIRA take much longer time. Solving a problem with $N = 120$ takes roughly the same time as it takes CSPF to solve a problem with $N = 500$.

All algorithms show exponential growth in their run time, since they roughly fall on a straight line in the log-log plot. This is quite surprising since these algorithms are polynomial in theory. We hypothesize that the large increase in run time could be because of the nature of these algorithms. For example, CSPF computes shortest paths for each traffic demand in turn. When a large number of traffic demands have been routed already, it becomes more difficult to find a new path for the new traffic demand as more candidate paths need to be pruned.

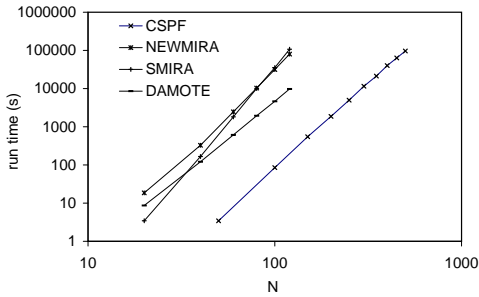


Fig. 4. Computation time for different algorithms.

Our result suggests that there is a lot of room to improve the computation efficiency of TE algorithms, especially when the network is large.

C. Evaluating objective 1: Maximizing the minimum free bandwidth

Several algorithms fit this objective, including CSPF when the metric is the inverse of the free capacity, NEWMIRA and SMIRA.

We first generate test problems with the optimal value δ set to 10000, which is large enough to accommodate all traffic demands. We set δ large to see how the algorithms perform without the capacity limit. In Fig. 5, we plot the results from CSPF, NEWMIRA and SMIRA.

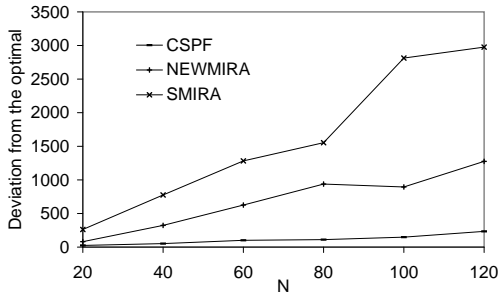


Fig. 5. Optimality gap for CSPF, NEWMIRA and SMIRA when $\delta = 10000$.

Surprisingly, elaborate algorithms, such as NEWMIRA and SMIRA, perform quite badly compared to a simple-minded algorithm such as CSPF. The optimality gap for NEWMIRA and SMIRA is considerably bigger than that for CSPF. One possible reason is that NEWMIRA and SMIRA try to even out the traffic early on, which leads to many traffic demands being routed over non-shortest paths. Non-shortest path routing would consume more bandwidth, leading to further deviation from the optimum.

We also generate problems with a smaller value of δ to see how the algorithms perform under the capacity limit. In Table I, we show the result for the NEWMIRA algorithm with $\delta = 500$. When the capacity limit is smaller, the algorithm indeed performs a little better because it is more aggressive at evening out the load to stay under the capacity limit. However, the capacity constraint is not able to force the algorithm to

TABLE I
OPTIMALITY GAP FOR NEWMIRA WITH DIFFERENT δ .

N	20	40	60	80	100	120
$\delta = 10000$	81	323	626	938	895	1276
$\delta = 500$	68	305	441	-	-	-

produce much better results, so that when N is large (≥ 80), no feasible solution could be found.

For both cases, the optimality gap grows as N increases. Even for our best performing algorithm CSPF, the optimality gap more than doubles when N doubles. This suggests that more algorithm research is needed to bridge the gap, especially when N is large. Note that we did not normalize the optimality gap result because it is not clear what it should be normalized against (N , or N^2 , or anything else?). Instead, we reported the absolute gap so that interested readers can derive any trend based on “normalized” results.

For dynamic MPLS TE problems, the sequence of traffic demand arrival obviously has a significant impact on the end result. To assess how much the sequence affects the end result, we generate several different sequences. The first is simply based on a random ordering.

The second sequence tries to create uneven traffic flow as new traffic demands arrive. To create such a sequence, we look at how traffic demands are routed in the optimal solution. We pick a link and put all traffic demands routed over it at the beginning of the sequence. Thus if an algorithm were to use the optimal routing, it would have fully loaded one link, but only lightly loaded other links after routing the first set of demands. We continue to pick links and append all remaining traffic demands on the link to the sequence.

The third sequence is the opposite. It tries to create even traffic flow as new traffic demands arrive. Again, we rely on the optimal solution to create such a sequence. We pick a link that is most loaded, then randomly choose one traffic demand routed on the link and append it to the sequence. This process repeats until all traffic demands are in the sequence. If an algorithm were to use the optimal routing, traffic on the links should be fairly even as new traffic demands are routed.

In Fig. 6, we show the results for CSPF algorithm with the various sequences. As shown, the difference in sequence can have a big impact on the result. A bad sequence could have twice the optimality gap as a good sequence.

D. Evaluating objective 2: Minimizing the maximum percentage utilization

Out of the algorithms we evaluate, DAMOTE is the only one that can optimize for objective 2. We use the score function as in equation (5) to achieve load balancing.

If we set the optimal result to be $\theta = 50\%$, we find that DAMOTE could not even find a feasible solution for any of our test problems. However, if we switch to a different algorithm, such as CSPF, for the same test problems, we can find a feasible solution. We believe DAMOTE is attempting

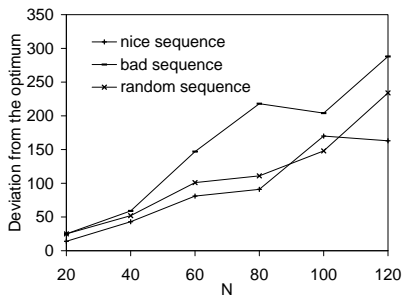


Fig. 6. Effect of traffic demands arrival sequence on the optimality gap. Based on CSPF algorithm.

TABLE II
THE OPTIMALITY GAP FOR OBJECTIVE 2 FOR DAMOTE.

N	20	40	60	80	100	120
Opt. gap	22.76%	6.09%	28.37%	17.83%	18.15%	16.31%

to even out the traffic at the very beginning and uses longer path to route traffic to achieve such goal. Hence, at the end, no capacity is left to route the remaining demands.

In Table II, we show the optimality gap for the DAMOTE algorithm. In order to get a feasible solution, we set the optimum to be $\theta = 20\%$. In the worst case, the optimality gap is 28%, which is more than double the optimal value. Even though the optimality gap expressed in terms of percentage is roughly consistent, the optimality gap expressed in terms of absolute bandwidth keeps on growing as N increases, which is consistent with our earlier observations.

E. Evaluating objective 4: Minimizing total delay

DAMOTE can optimize for the total delay if we set the score function to be equation (3). The results are shown in Table III. For ease of comparison, we normalize the result and report the delay as $\sum_i \frac{c_i}{c_i - l_i}$. Recall from our construction method that all c_i are equal, therefore, the normalization does not affect the end result.

Again, we see the same trend that the optimality gap grows rapidly as N increases. For the network with $N = 120$, DAMOTE completely saturates at least one link and several traffic demands could not be accommodated. Because some links are 100% utilized, the total delay is ∞ .

V. CONCLUSION AND FUTURE WORK

TE algorithms play an important role in helping the ISPs to more efficiently utilize their existing network infrastructure.

TABLE III
THE OPTIMALITY GAP IN TOTAL DELAY FOR DAMOTE.

N	20	40	60	80	100	120
optimum	100	200	300	400	500	600
DAMOTE	136.92	351.97	624.82	1842.79	2147.53	∞

Since many TE problems are NP-hard, it is important to devise efficient heuristic algorithms. Unfortunately, there is not a comprehensive methodology to evaluate these algorithms. As a result, it is not only unclear which algorithm is better, but also unclear how much room there is for improvement. We propose a novel methodology where we construct test instances with known optimal results. By comparing the known optimum with the output of the heuristics, we can reveal the optimality gap. Using our methodology, we show that the result from TE algorithms are quite far away from the optimum and the optimality gap grows rapidly as the problem size increases. Our study suggests that much more algorithm research is needed in order to bridge the optimality gap. Beyond algorithm evaluation, our methodology is also an invaluable debugging tool. Our methodology has been applied at one large ISP to uncover several implementation bugs.

There are a number of directions for our future work. We have only evaluated a few TE algorithms, which is a small fraction of the existing work. It is beneficial to use our methodology to systematically evaluate a larger set of TE algorithms. The construction method could also benefit from improvements. We have to set the link capacities to guarantee that we know the optimal solution. Designing a new construction method that can accommodate arbitrary link capacities would provide better coverage.

REFERENCES

- [1] B. Fortz and M. Thorup, "Internet traffic engineering by optimizing OSPF weights," in *Proc. Infocom*, 2000, pp. 519–528.
- [2] —, "Increasing Internet capacity using local search," *Computational Optimization and Applications*, vol. 29, no. 1, 2004.
- [3] M. Kodialam and T. Lakshman, "Minimum interference routing with applications to mpls traffic engineering," in *Proc. Infocom*, 2000.
- [4] B. Wang, X. Su, and C. Chen, "A new bandwidth guaranteed routing algorithm for MPLS traffic engineering," in *Proc. IEEE International Conference on Communications (ICC)*, 2002.
- [5] I. Iliadis and D. Bauer, "A new class of online minimum-interference routing algorithms," in *Proc. Networking*, 2002.
- [6] P. V. Mieghem, H. D. Neve, and F. Kuipers, "Hop-by-hop quality of service routing," *Computer Networks*, vol. 37, no. 3-4, 2001.
- [7] P. V. Mieghem and F. Kuipers, "Concepts of exact quality of service algorithms," *IEEE/ACM Trans. Netw.*, 2004.
- [8] L. Fratta, M. Gerla, and L. Kleinrock, "The flow deviation method: An approach to store-and-forward communication network design," *Networks*, vol. 3, pp. 97–133, 1971.
- [9] G. Leduc, H. Abrahamsson, S. Balon, S. Bessler, M. D'Arienzo, O. Delcourt, J. Domingo-Pascual, S. Cerav-Erbas, I. Gojmerac, X. Masip, A. Pescape, B. Quoitin, S. P. Romano, E. Salvadori, F. Skivee, H. T. Tran, S. Uhlig, and H. Umit, "An open source traffic engineering toolbox," *Computer Communications*, vol. 29, no. 5, pp. 593–610, March 2006.
- [10] B. M. Waxman, "Routing of multipoint connections," *IEEE J. Sel. Areas Commun.*, vol. 6, no. 9, Dec. 1988.
- [11] A. Medina, A. Lakhina, I. Matta, and J. Byers, "Brite: An approach to universal topology generation," in *Proceedings of the International Workshop on Modeling, Analysis and Simulation of Computer and Telecommunications Systems- MASCOTS'01*, 2001.