International Scholarly Research Network
ISRN Communications and Networking
Volume 2012, Article ID 914953, 9 pages
doi:10.5402/2012/914953

Research Article

Optimizing Virtual Private Network Design Using a New Heuristic Optimization Method

Hongbing Lian and András Faragó

Erik Jonsson School of Engineering and Computer Science, University of Texas at Dallas, Richardson, TX 75083, USA

Correspondence should be addressed to Hongbing Lian, lian0716@yahoo.com

Received 21 March 2012; Accepted 19 April 2012

Academic Editors: A. Maaref, C. Pomalaza-Ráez, and A. Zanella

In virtual private network (VPN) design, the goal is to implement a logical overlay network on top of a given physical network. We model the traffic loss caused by blocking not only on isolated links, but also at the network level. A successful model that captures the considered network level phenomenon is the well-known reduced load approximation. We consider here the optimization problem of maximizing the carried traffic in the VPN. This is a hard optimization problem. To deal with it, we introduce a heuristic local search technique called landscape smoothing search (LSS). This study first describes the LSS heuristic. Then we introduce an improved version called fast landscape smoothing search (FLSS) method to overcome the slow search speed when the objective function calculation is very time consuming. We apply FLSS to VPN design optimization and compare with well-known optimization methods such as simulated annealing (SA) and genetic algorithm (GA). The FLSS achieves better results for this VPN design optimization problem than simulated annealing and genetic algorithm.

1. Introduction

In the VPN setting the goal is to implement a logical overlay network on top of a given physical network. We consider here the optimization problem of maximizing the carried traffic in the VPN. In other words, we want to minimize the loss caused by blocking some of the offered traffic, due to insufficient capacity in the logical links.

A key feature in the VPN setting is that the underlying physical network is already given. Thus, our degree of freedom lies only in dimensioning the logical (virtual) links. However, since the given physical link capacities must be obeyed and a physical link may be shared by several logical links, we can reduce the blocking on a logical link possibly only by taking away capacity from other logical links. Therefore, we may be able to improve a logical link only by degrading others. The above described situation leads to a hard optimization problem.

Mitra et al. [1] analyzed a network loss probability caused by blocking with *fixed point equations* (FPEs). They derived the loss probability only based on assumption of link independence. Actual difficulty is posed by the fact that we need to model the traffic loss caused by blocking not only on isolated links, but also at the network level. This means we also need to take into account that the loss suffered on a link reduces the offered traffic of other links and vice versa, so a complex system of mutual influences arise. This situation calls for a more sophisticated machinery than blocking formulas (such as Erlang's formula) that compute the blocking probability only for a single link viewed in isolation. A successful model that captures the considered network level phenomenon is the reduced load approximation. We review it in the next section so that we can then use it in our VPN design model.

In this paper we investigate a virtual private network (VPN) design problem. We adopt a complex model to describe the carried traffic [2–4]. To deal with the arising hard optimization problem, we use a new heuristic local search technique called landscape smoothing search (LSS) proposed by Lian and Faragó, authors of this paper in

[5]. This study first describes the LSS heuristic method and then we modify the original LSS method to a fast landscape smoothing search (FLSS) method to overcome the slow search speed for the case when the objective function calculation is very time consuming. We apply FLSS to VPN design optimization and compare with existing methods such as simulated annealing (SA) [6, 7] and genetic algorithm (GA) [8, 9].

Basically this study consists of two parts. The first part is the proposal and the analysis of carried traffic for virtual private network (VPN). In the second part we propose the landscape smoothing search (LSS) [10] method and the fast LSS (FLSS) heuristic method and apply them to VPN optimization.

The remainder of the paper is organized as follows: Section 2 presents a reduced load approximation to model to capture the VPN carried traffic. Section 3 analyzes a nonlinear network level optimization model. The last part of Section 3 also provides the carried traffic objective function for VPN design optimization. Section 4 presents initial results with the original Landscape Smoothing Search (LSS). Section 5 presents Fast Landscape Smoothing Searching (FLSS). Section 6 presents numerical optimization results for VPN optimization and discussion of the features of three heuristic FLSS, SA and GA. Finally, Section 7 concludes the paper.

2. Reduced Load Approximation

The principle of this approach is "folklore" in traffic engineering and had been presented already in the 1960's by Cooper and Katz [11]. Nevertheless, in-depth exact investigation was done only much later, in the papers by Kelly [2] and Whitt [4]. For a comprehensive exposition of related results see the book of Ross [3].

To present the most fundamental case, let us consider a network of J links. A general link will be denoted by j, that is, we index the edges of the network graph here, rather than the nodes. Link j has capacity C_j. Let us assume that a set R of fixed routes is given in the network. A route $r \in R$, in general, can be an arbitrary subset of the link set. Here we do not need the assumption that it is a path in the graph theoretic sense. Of course, the practically most important case is when it is actually a path. There may be several routes between the same pair of nodes, even on the same sequence of links. The offered traffic V_r (the demand) to a given route $r \in R$ arrives as a Poisson stream and the streams belonging to different routes are assumed to be independent.

The incidence of links and routes is given by a matrix $= [A_{jr}]$, $j = 1, \ldots, J, r \in R$. If link j is on route r, then $A_{jr} = 1$, otherwise $A_{jr} = 0$. The call holding times are independent random variables, and the holding periods of calls on the same route are identically distributed with finite mean. However, this distribution can otherwise be arbitrary. The central approximation assumption of the model is that the blocking of different links are probabilistically independent events. Let us denote the blocking probability of link j by B_j. The reduced load approximation says that the Poisson stream is thinned by a factor of $(1 - B_j)$ on each traversed link independently. Hence the carried traffic on route r can be expressed as

$$V_r \prod_{j=1}^{J} \left(1 - B_j\right)^{A_{jr}}., \tag{1}$$

Note that the factor $(1 - B_j)^{A_{jr}}$ is 1 if the line is not traversed by the route (because then $A_{jr} = 0$); this is why the product can be taken for all links without taking care of which links are traversed by the route.

The carried traffic on link j is obtained if we sum up the carried traffic of all routes that traverse the link:

$$\sum_{r \in R} A_{jr} V_r \prod_i (1 - B_i)^{A_{ir}}. \tag{2}$$

Again, the summation is simply extended for all routes since $A_{jr} = 0$ holds for those that do not contain link j, making their contribution disappear from the sum.

If the total *offered load* to link j is denoted by ρ_j, then (2) should be equal to $\rho_j(1 - B_j)$, since the latter is the carried traffic on link j, obtained by thinning the offered load by the factor $1 - B_j$. Thus, we can write the equation

$$\rho_j \left(1 - B_j\right) = \sum_r A_{jr} V_r \prod_i (1 - B_i)^{A_{ir}}, \tag{3}$$

or, after canceling the factor $(1 - B_j)$

$$\rho_j = \sum_r A_{jr} V_r \prod_{i \neq j} (1 - B_i)^{A_{ir}}. \tag{4}$$

Further equations can be obtained by using that B_j depends on ρ_j and C_j in this model via Erlang's formula:

$$B_j = E\left(\rho_j, C_j\right) = \frac{{\rho_j}^{C_j}/C_j!}{\sum_{i=0}^{C_j} \rho_j^i / i!}. \tag{5}$$

(Note: in the case C_j is not an integer, we can use an analytic continuation of Erlang's formula, see [12].)

Writing out (4) and (5) for all $j = 1, \ldots, J$, we obtain a system of $2J$ equations for the $2J$ unknown quantities $\rho_j, B_j, j = 1, \ldots, J$. We can observe that B_j can be computed from (5) directly, once the values of the ρ_j variables are known (the link capacities are given). Therefore, the core of the problem is to compute $\rho_j, j = 1, ..., J$. Eliminating B_j from (4) by (5), we obtain a system of equations directly for the ρ_j variables:

$$\rho_j = \sum_r A_{jr} V_r \prod_{i \neq j} \left(1 - E\left(\rho_j, C_j\right)\right)^{A_{ir}}. \tag{6}$$

This system of equations (or, equivalently, the systems (4) and (5) together) is called *reduced load approximation*.

Alternatively, the equations are also called the *Erlang fixed point equations*.

The concept of *fixed point* comes into the picture in the following way. Let us use a vector notation $\rho = [\rho_1, \ldots, \rho_J]$ and define a function f: $R_+^J \to R_+^J$ by

$$f_j(\rho) = [f_1(\rho), \ldots, f_J(\rho)], \tag{7}$$

where $f_j(\rho)$, $j = 1, \ldots, J$ is given as

$$f_j(\rho) = \sum_r A_{jr} V_r \prod_{i \neq j} \left(1 - E\left(\rho_j, C_j\right)\right)^{A_{ir}}. \tag{8}$$

Now the system (6) can be compactly formulated as

$$\rho = f(\rho). \tag{9}$$

In other words, we have to find a fixed point of the mapping f: $R_+^J \to R_+^J$.

There are some natural questions that arise here immediately. Does a solution (a fixed point) always exist? If one exists, is it unique? How can we find it algorithmically in an efficient way? The fundamental theorem characterizing this model was proven by Kelly [2] (see also [13]). We also outline its proof, since the proof contains some concepts that we are going to use later.

Theorem 1. *The Erlang fixed-point equations always have a unique solution.*

Proof. The existence of the solution follows from the fact that the function f, defined above, is a continuous mapping of the closed J-dimensional unit cube $[0, 1]^J$ into itself, therefore by the well-known Brouwer fixed point theorem it has a fixed point. (Brouwer's fixed-point theorem says that any continuous function that maps a compact convex set into itself always has a fixed point.)

To show the uniqueness of the fixed point, we define an auxiliary function $U(y, C)$ in a tricky way, by the implicit relation

$$U\left(-\log(1 - E(v, C)), C\right) = v(1 - E(v, C)), \tag{10}$$

where $E(v, C)$ is Erlang's formula. The interpretation of $U(y, C)$ is that it is the average number of circuits in use (the utilization) on a link of capacity C when the blocking probability is $1 - e^{-y}$. In other words, $U(y, C)$ measures the link utilization as a function of a logarithmically scaled blocking probability $y = -\log(1 - B)$.

Define now an optimization problem, as follows:

$$\begin{aligned} &\text{minimize} \quad \sum_r V_r \exp\left(-\sum_j y_j A_{jr}\right) + \sum_j \int_0^{y_j} U\left(z, C_j\right) dz, \\ &\text{subject to} \quad y_j \geq 0, \quad j = 1, \ldots, J. \end{aligned} \tag{11}$$

The first sum in objective function is a strictly convex function. Since $U(y, C)$ is strictly increasing, therefore, the integrals in the second sum are also strictly convex. Hence the above optimization problem, being the minimization of a strictly convex function over a convex domain, has a unique minimum. Consider now the stationary equations obtained by equating the derivative of the objective function with zero:

$$\sum_r V_r \exp\left(-\sum_j y_j A_{jr}\right) = U\left(y_j, C_j\right), \quad j = 1, \ldots, J. \tag{12}$$

Using the definition of $U(y, C)$ and applying the transformation $B_j = 1 - e^{-y_j}$, we get back precisely the Erlang fixed-point equations from (12). Since we already know that there is a nonnegative solution to the fixed point equations, this implies that the stationary equations (12) also have a nonnegative solution, which is thus the minimum of the optimization problem (11). Conversely, each solution of (11) corresponds to a fixed point through the transformation $B_j = 1 - e^{-y_j}$. Since by the strict convexity there is a unique minimum, therefore (12) cannot have another solution, which implies the uniqueness of the fixed point, thus completing the proof. □

Having proved the existence and uniqueness of the fixed point, a natural question is how to find it algorithmically. The simplest algorithm is to do iterated substitution using the function defined in (7), (8). We can start with any value, say $\rho^{(0)} = [1, \ldots, 1]$ and then iterate as

$$\rho^{(i+1)} = f\left(\rho^{(i)}\right), \tag{13}$$

until $\rho^{(i+1)}$ and $\rho^{(i)}$ are sufficiently close to each other. This method works very well in practice, although convergence is not guaranteed theoretically, since $f(\cdot)$ is not a contraction mapping, that is, $||f(x) - f(y)|| < \alpha||x - y||$ does not necessarily hold for some constant $\alpha < 1$. In fact, there exist examples for nonconvergence, see Whitt [4].

Another algorithmic possibility is solving the convex programming problem (11). Although this is guaranteed to work, nevertheless, it offers a much more complicated algorithm, which is made even worse by the implicit definition of the function $U(y, C)$. Therefore the practical algorithm is the iterated substitution, even though it is not guaranteed to converge in pathological cases.

The presented model is the base case, when routing is fixed and traffic is homogeneous. Various extensions exist for more complicated cases, see for example, [3, 14]. Unfortunately, they lack the nice feature of the unique fixed point. In the next section, extensions to heterogeneous traffic will be used for cases when the reduced load approximation is embedded into optimization models.

3. A Nonlinear Network Level Optimization Model

In this section we build a nonlinear network level optimization model based on the reduced load approximation. Recall

that we considered the situation when logical (virtual) subnetworks exist on top of the given physical network. They are realized by logical links. A logical link is, in general, a subset of the physical links. It can be, for example, a route in the physical network. Our objective is to allocate capacity to the logical links such that the physical capacity constraints are obeyed on every physical link and the total carried network traffic is maximized. Note that since the logical links share physical capacities. Therefore if we want to decrease blocking on a logical link by giving more capacity, we can only do this by taking away capacity from others, thus degrading other logical links. It is intuitively clear that an optimization problem arises from this VPN design.

Since the model is built on the reduced load approximation (Section 2), therefore we use the same notation. The network contains J logical links, labeled $1, 2, \ldots, J$. The capacity of logical link j is C_j. Since logical link capacities are not fixed in advance (we want to optimize with respect to them!), therefore the C_j are variables. Let $C = (C_1, C_2, \ldots, C_J)$ be the vector of logical link capacities.

The condition that the sum of logical link capacities on the same physical link cannot exceed the physical capacity can be expressed by a linear system of inequalities. Let C^{phys} be the vector of given physical link capacities. Furthermore, let S be a matrix in which the jth entry in the ith row is 1 if logical link j needs capacity on the ith physical link, otherwise 0. Then the physical constraints can be expressed compactly as $SC \le C^{\text{phys}}$.

A set R of fixed routes is given in the network. A route is a sequence of logical links. There may be several routes between the same pair of nodes, even on the same sequence of logical links. The offered traffic (the demand) to a given route $r \in R$ is V_r and is assumed to arrive as a Poisson stream. The streams belonging to different routes are assumed independent. Holding times are independent of each other and holding periods of sessions on the same route are identically distributed.

We consider heterogeneous (multirate) traffic. To preserve the nice properties of the Erlang fixed-point equations, we adopt the following homogenization approach: a session (call) that requires b units of bandwidth is approximated by b independent unit bandwidth calls.

The capacity (bandwidth) that a session on route r requires is denoted by b_{jr} on link j (for the sake of generality, we allow that it may be different on different links). If the route does not traverse link j, then $b_{jr} = 0$. Note that b_{jr} plays the same role here as A_{jr} in the description of the reduced load approximation, but b_{jr} can now also take values other than 0 and 1.

According to the applied approximations, the total carried traffic in the network is expressed as

$$\sum_r V_r \prod_j \left(1 - B_j\right)^{b_{jr}}, \tag{14}$$

where B_j is the (yet unknown) blocking probability of logical link j. ($j = 1, 2, \ldots, J$).

Our objective is to find the vector C of logical link capacities, subject to the physical constraints $SC \le C^{\text{phys}}$ and $C \ge 0$, such that the total carried traffic is maximized:

$$\begin{aligned} &\text{maximize} \quad \sum_r V_r \prod_j \left(1 - B_j\right)^{b_{jr}}, \\ &\text{subject to} \quad SC \le C^{\text{phys}}, \quad C \ge 0, \end{aligned} \tag{15}$$

where C_j are variables and the dependence of B_j on C_j is defined by the Erlang fixed-point equations:

$$B_j = E\left(\left(1 - B_j\right)^{-1} \sum_{r \in R} b_{jr} V_r \prod_{i \ne j} (1 - B_i)^{b_{ir}}, C_j\right), \tag{16}$$

where $j = 0, 1, \ldots, J$.

4. Heuristic Methods to Solve Network Design Problems

This section presents heuristic methods to solve optimization problems, including network design tasks. Here we only focus on combinatorial optimization algorithms for problems of the following form:

$$\text{minimize} \quad F(x) \text{ for } x \in S, \tag{17}$$

where S is a very large finite set of feasible points (solution space, optimization space). The variable x can take different forms: it can be a binary string, an integer vector, or mixed integer and label combination.

4.1. Well-Known Heuristic Methods. One of the well-known general stochastic methods is *simulated annealing* (SA) [7, 9, 15]. As a local search method, SA also uses the notion of neighborhood, but applies randomness to choosing the next step to avoid getting trapped in local optima. We can refer [5, 7, 9] for SA pseudocode.

Another well-known heuristic method is *genetic algorithm* (GA). The genetic algorithm (GA) is also often applied in combinatorial optimization problems [5, 6, 8, 9]. The procedure first selects parent solutions for generating offspring. Then it performs two basic procedures: crossover (x) with probability P_c and mutation (x) with probability P_m. We can refer [5, 6, 8, 9] for GA pseudocode. Genetic algorithm does not use local search. It may jump away from the best point even when it is very close to it.

4.2. Description of Landscape Smoothing Search (LSS). Our proposed LSS is a general optimization technique suitable for a wide range of combinatorial optimization problems. The authors of this paper proposed the original LSS to solve call admission control optimization problem for cognitive radio network in [5]. To better understand the idea of LSS we give an introduction to its basic form here. LSS can get out of local optima, and effectively conduct local search in the vast search spaces of hard optimization problems. The key idea is that we continuously change the objective/fitness function of the

Procedure *algorithm LSS* (F,λ,X,G)
begin
create initial feasible solution X_0
set $d_h(X)= 0$, $(h = 1,\ldots,H)$
(set all initial smoothing factors to zero)
  for $i: = 1$ **to** max-iterations
  begin
    $F'(X) = F(X) + \sum_h [\lambda_h^* d_h(X)]$
    X_i = Local search $(F', \lambda_h, X_{i-1}, G)$
    for each $h: = 1$ **to** H
    begin
      adjust $d_h(X)$;
      (X_h is the local optimum point)
      if $F(X_i) < F^{\text{best}}$ **then**
        $F^{\text{best}} : = F(X_i)$;
        $X^{\text{best}} : = X_i$;
      end
      adjust λ_h for adaptive λ;
    end
  end procedure
  LSS returns X^{best} where $F(X^{\text{best}})$ is the minimum of all solutions so far.

Algorithm 1: Landscape smoothing search (LSS).

problem to be minimized with a set of smoothing functions that are dynamically manipulated during the search process to steer the heuristics to get out of the local optima.

The objective function $F(X)$ is extended with a smoothingfunction $S(X)$ as follows:

$$F'(X) = F(X) + S(X) = F(X) + \sum_h \lambda_h \cdot d_h(X), \tag{18}$$

where $S(X)$ is the smoothing function:

$$S(X) = \sum_h \lambda_h \cdot d_h(X), \tag{19}$$

which contains "landscape smoothing functions." We define them as $d_h(X_h) = 0$, if we hit this local optimal point X_h for the first time. $d_h(X_h) = d_h(X_h) + 1$,if we hit this local optimal point X_h for the second or more time. λ_h is the smoothing step constant and it may be adaptively changed.

We also record the best point reached so far, and the local optima during the search. Every time we hit a local optimum, we compare it with the global optimum (best point so far). Then we adjust the landscape smoothing factors to let the local search get away from the local trap.

As we keep changing the objective function, we gradually "smooth out" the landscape to get rid of the local holes that trap the search. We never fill a hole, however, before the second time we reach it. This way the search trace never misses a hole that could be the global optimum. See Figure 1.

We assume F is the objective function; λ is a group of constants that serve as a landscape smoothing step factors, which can be constants or can also be adaptively changed as λ_h. G represents the problem specification (e.g. network topology, demand matrix or some constraints). We give the pseudo code here for landscape smoothing search (LSS) in Algorithm 1.

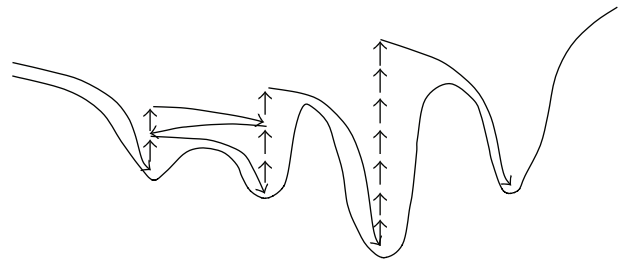

Figure 1: Landscape smoothing search (LSS).

5. Initial Results with Original Landscape Smoothing Search (LSS)

In this section we demonstrate a sample non-linear hard VPN design problem and the optimization results achieved with LSS and simulated annealing (SA).

5.1. A Sample VPN. We use a VPN with 40 virtual links and 12 routes shown as in Figure 2 for simulation. For example, $e_2 = c_0 + c_8$ means physical link e_2 consists of two virtual links c_0 and c_8, $e_9 = c_{12} + c_{27} + c_{35}$ means physical link e_9 consists of three virtual links c_{12},c_{27}, and c_{35}, dashed route $r_2 = c_8 + c_9 + c_{10}$ means route r_2 consists of three virtual links c_8, c_9, and c_{10}, dotted route $r_5 = c_{18} + c_{19} + c_{20} + c_{21}$.

5.2. Initial Results with LSS. We use the sample VPN described in Section 5.1. The diagram in Figure 3 shows initial results with landscape smoothing search (LSS) and simulated annealing (SA).

Our optimization results were obtained through a PC with AMD Sempron 2500+ CPU. The searching time is the

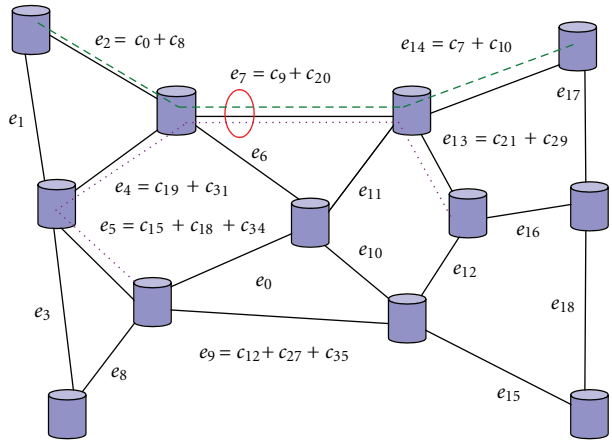

Figure 2: A sample VPN structure.

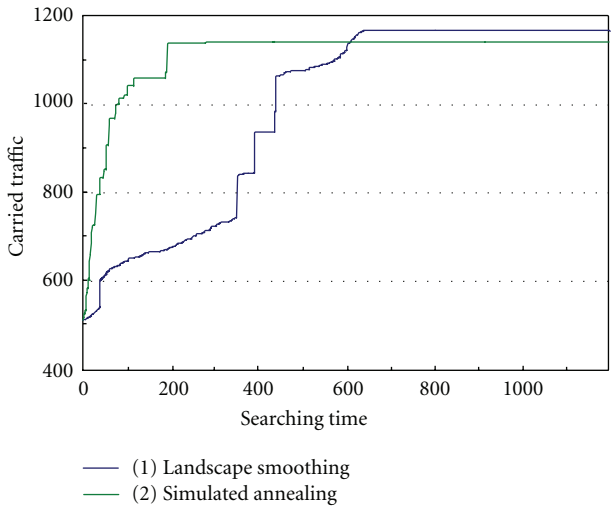

Figure 3: VPN design search trace with LSS and SA.

time needed to obtain best solution so far. The searching time unit is 10 seconds. For each heuristic method, these measures were recorded each time there was an improvement in the best carried traffic value. So in each point of the search curve, the X-axis value is the searching time in seconds and the Y-axis value is the best carried traffic value we achieve so far. We find that LSS gives better result in long time, but the search speed is slower than SA. When the objective function takes very long time to calculate we find that the original landscape smoothing searching (LLS) heuristic method is slower than simulated annealing (SA). As we know SA makes a possible move with a random neighbor to find a better objective value. The original LSS makes a possible move by picking the best neighbor of all direct neighbors. If for a case which has N ($N = 40$) direct neighbors it needs to do N ($N = 40$) objective function calculations to make a possible move. The objective function calculation of VPN design itself is a complex iterative process using (16). It may take 50 to 90 iterations to calculate an accurate value. So the objective function calculation of VPN design is a time consuming

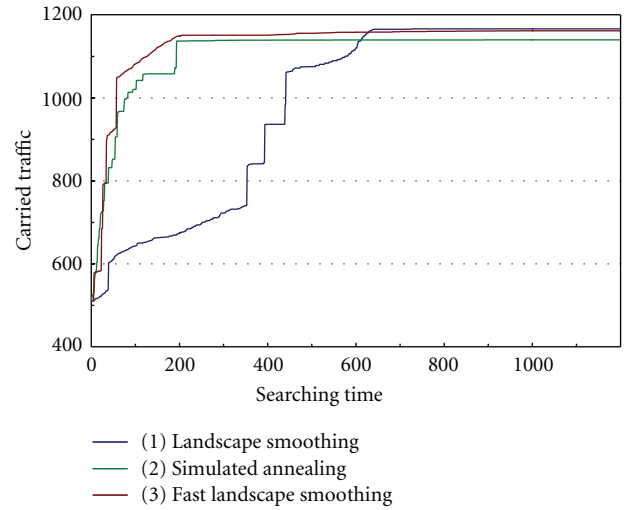

Figure 4: VPN Search trace with fast LSS and SA.

process. This explains why the original LSS method is slower than SA here for VPN design case.

6. Fast Landscape Smoothing Search (FLSS)

To improve the search speed of LSS, we modify the original LSS into a fast landscape smoothing search (FLSS). The FLSS has two (or more) phases of search. The first phase we call rough search phase. In the rough search phase we divide the set of N neighbors into several subsets, like $N1$, $N2$, $N3$, and $N4$. We apply the LSS method on subneighborhoods one at a time, and also we apply LSS on subneighbor one by one. This way we make every LSS move with much fewer objective function calculations. The second phase we call fine search phase. In the fine search phase we use the original LSS method on the whole neighborhood, as in the original LSS. This way we will not miss the possible global optima in the process we search all N direct neighbor. The improved result is shown in Figure 4.

7. More Numerical and Optimization Results

This section presents more numerical and optimization results of carried traffic load for the VPN design problem.

7.1. Optimization Results with Simulated Annealing (SA). Figure 5 shows simulated annealing (SA) optimization search traces for function (15) with 3 different T-reduce values (cooling speed factor). We can see SA convergence speed may be affected by setting the temperature reduce cooling value. At the beginning of the SA search procedure, SA converges very fast. Then SA convergence becomes very slow. After a while it takes a long time for SA to find the next better value.

Figure 6 shows simulated annealing (SA) optimization search trace with different initial settings. In all these figures, legend initial settings like [Initial-2 = 4 3 2 1 4 3 2 1] means virtual link capacities

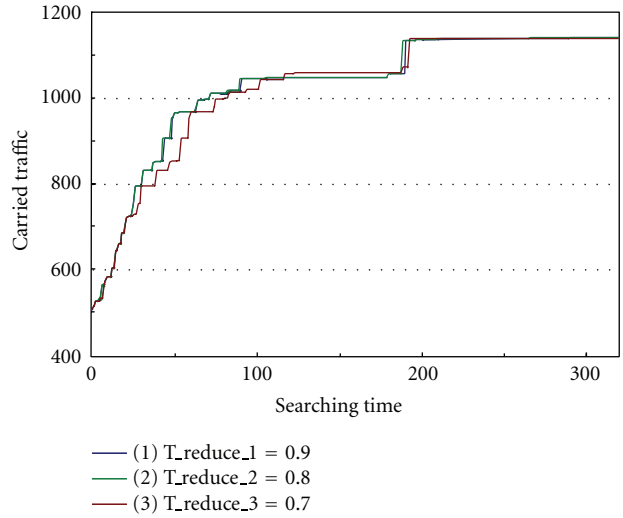

Figure 5: Simulated annealing search trace with different T-reduce values.

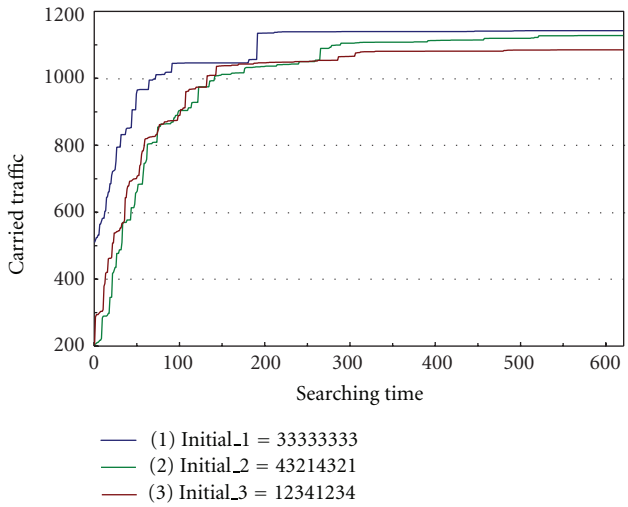

Figure 6: Simulated annealing search trace with different initial settings.

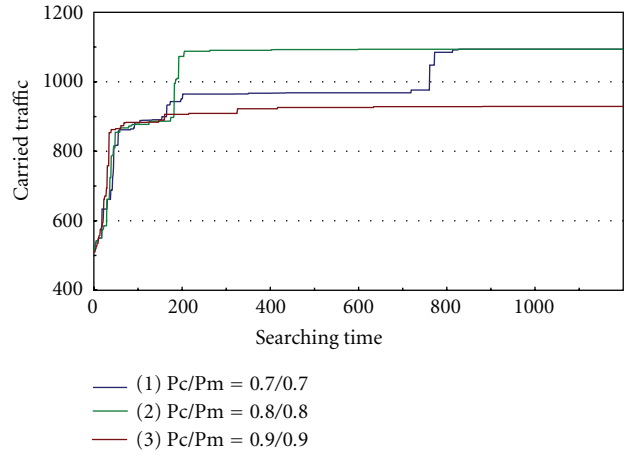

Figure 7: Genetic algorithm search trace with different P_c/P_m values.

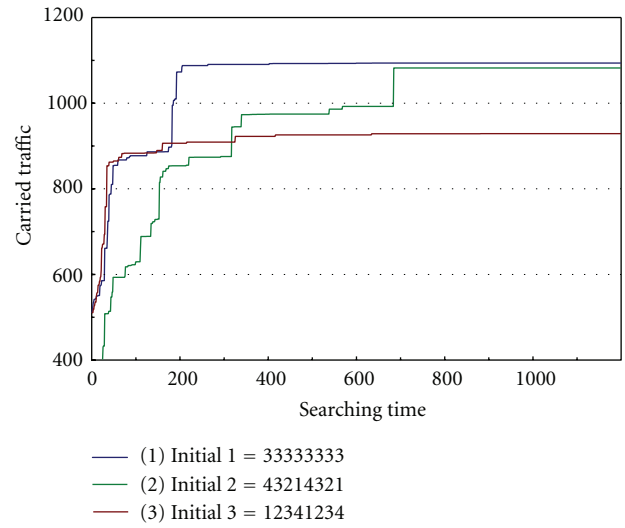

Figure 8: Genetic algorithm search trace with different initial settings.

$C_j[0], C_j[1], C_j[2], C_j[3], C_j[4], C_j[5], C_j[6], C_j[7], \ldots =$ 4 3 2 1 4 3 2 1,…. From Figure 6 we can see the SA optima value depends on the initial settings. Different initial settings will lead to a slightly different SA optima value.

7.2. Optimization Results with Genetic Algorithm (GA). Figure 7 shows genetic algorithm (GA) optimization search traces for function (16) with 3 sets of different P_c/P_m values. P_c is the probability for crossover. P_m is the probability for mutation. From Figure 7 we can see GA convergence speed may be affected by settings of crossover/mutation probability values. At the beginning of the GA search procedure, GA converges very fast. Then GA converges very slowly. After a while it takes a long time for GA to find the next better value.

Figure 8 shows Genetic Algorithm (GA) optimization search trace with different initial settings. From Figure 8 we can see the GA near optima value depends on the initial settings. Here the notation like [Initial-2 = 1 2 3 4 1 2 3 4] means virtual link capacities $C_j[0]$, $C_j[1]$, $C_j[2]$, $C_j[3]$, $C_j[4]$, $C_j[5]$, $C_j[6]$, $C_j[7]$, … = 1 2 3 4 1 2 3 4,…. Different initial settings will lead to slightly different GA optima values. This kind of search trend is similar to that which we see in SA.

7.3. Optimization Results with Fast Landscape Smoothing Search (FLSS). Figure 9 shows fast landscape smoothing search (FLSS) optimization search trace with different smoothing step factor λ values. From Figure 9 we can see LSS convergence speed may slightly affected by setting of

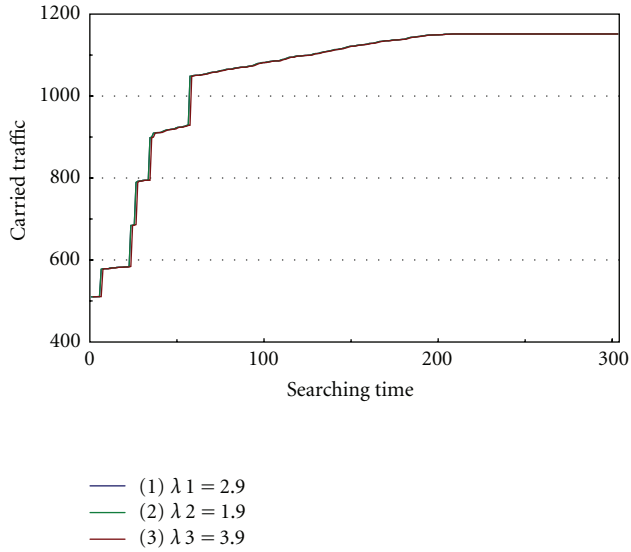

Figure 9: Fast landscape smoothing search trace with different λ values.

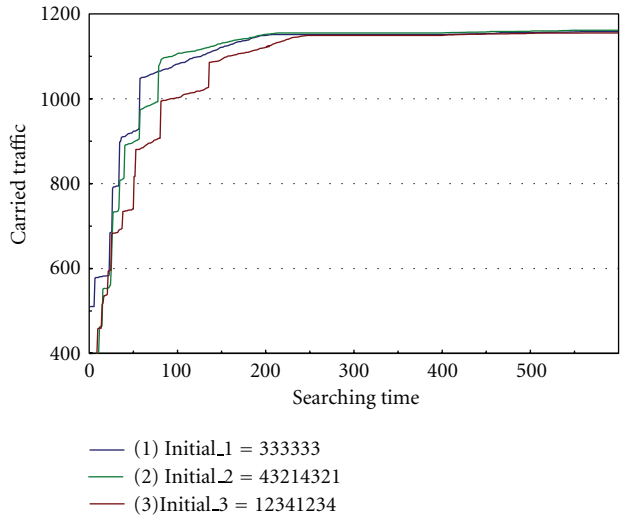

Figure 10: Fast landscape smoothing search trace with different initials.

smoothing step factor λ value. FLSS converges relatively fast and stabilizes smoothly.

Figure 10 shows Fast Landscape Smoothing Search (FLSS) optimization search trace with different initial settings. Here the notation [Initial-3 = 1 2 3 4 1 2 3 4] means virtual link capacities $C_j[0]$, $C_j[1]$, $C_j[2]$, $C_j[3]$, $C_j[4]$, $C_j[5]$, $C_j[6]$, $C_j[7]$, ... = 1 2 3 4 1 2 3 4,.... From Figure 10 we can see the FLSS optima value depends on the initial settings. Different initial settings will lead to slightly different optima values.

These results show that FLSS is much less sensitive to the adjustable parameters than SA and GA. And FLSS is also much less sensitive to the initial settings of search starting point. With different initial settings, FLSS converges to three

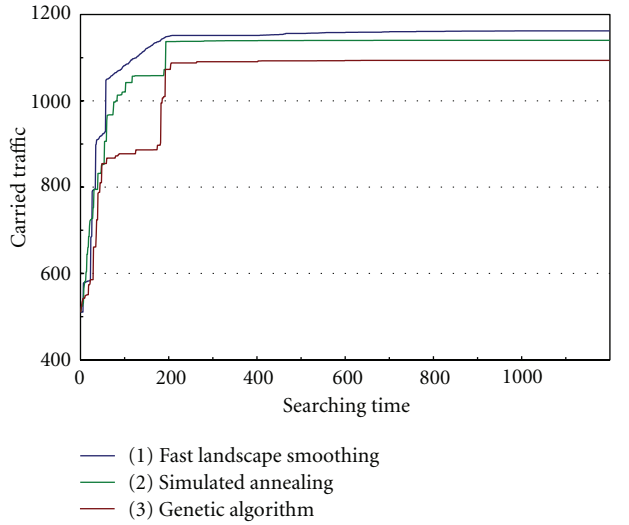

Figure 11: Three optimization methods—search results comparison.

near optima values that are very close. The difference of the three values is almost negligible.

7.4. Three Optimization Methods Comparison. To compare the three optimization methods, we put fast landscape smoothing search (FLSS), simulated annealing (SA), and genetic algorithm (GA) optimization search traces together in Figure 11.

Here the three methods all use the same initial settings of [3 3 3 3 3 3 3 3]. FLSS uses smoothing step factor λ of 2.90. SA uses cooling T-reduce factor of 0.7. GA uses crossover/mutation probabilities P_c/P_m of 0.9/0.9. We can see that FLSS starts slower than SA and GA at the beginning of the search, but FLSS catches up later and achieves a better objective function value.

We can see that SA and GA are not likely to get better results after time unit of 250. FLSS keeps getting better results after time unit of 250 and even 500. In this VPN case, FLSS slightly over performs SA and GA in the long run. So FLSS is a very good candidate for heuristic methods.

The results of these heuristic methods are case dependent. Based on the three network design cases we used, we compare these heuristic method features in Table 1 of the following page.

LSS and FLSS use adaptive smoothing function and they reach the optima by finding the best neighbor. SA jumps randomly with reduced cooling probabilities and reaches the optimum by chance. GA uses crossover and mutation with P_c/P_m probabilities and reaches the optimum by chance. SA and GA might jump away from the global optimum even when they are very close to it.

8. Conclusion

In this paper, we model the VPN traffic loss caused by blocking not only on isolated links, but at the network level.

Table 1: Four heuristic method feature comparison.

Method	Neighbor local search used?	Random?	Trap escaping mechanism	Search speed
Simulated annealing (SA)	Yes Use with probability Use random neighbor	Yes	Randomly jump with probability, controlled by cooling schedule. (Reach optima by chance)	Fast
Genetic algorithm (GA)	No	Yes	Use crossover, mutation with probability to jump out. (Reach optima by chance)	Slow
Landscape smoothing search (LSS)	Yes Use best neighbor	No	Use adaptive smoothing function. (Reach optima by best neighbor)	Medium
Fast landscape smoothing search (FLSS)	Yes Use best neighbor	No	Use adaptive smoothing function. (Reach optima by best neighbor)	Fast

We take into account that the loss suffered on a link reduces the offered traffic of other links and vice versa. We formulate the VPN design problem as the optimization problem of maximizing the carried traffic in the VPN. So that the VPN optimization becomes a hard optimization problem. We used a heuristic method called landscape smoothing search (LSS) and applied it to this problem. We find that LSS can get better result than SA but with a slower search speed. The reason is that in this VPN case, the objective function calculation of VPN carried traffic is a very time consuming process. To improve the speed of the original LSS we proposed a new fast landscape smoothing (FLSS) method.

The slow search speed drawback of the original LSS is overcome in the FLSS. Our FLSS method is also compared with popular heuristic methods such as simulated annealing (SA) and genetic algorithm (GA). We find the FLSS technique to be simple to implement. The three techniques were tested in many experiments with different VPN initial settings and different adjustable parameters to compare the optimization performance. The results show that the FLSS is much less sensitive to the adjustable parameters than SA and GA are. The FLSS is also less sensitive to the initial settings of search starting point than SA and GA are. With different initial settings, the FLSS converges to almost the same near optima value each time. The overall results show that FLSS outperforms the SA and the GA techniques both in terms of solution quality and optimization speed. Therefore based on these results, the fast landscape smoothing search (FLSS) technique can be strong candidate in solving hard optimization problems in network design.

Acknowledgment

The authors are grateful for the support of NSF Grant CNS-1018760.

References

[1] D. Mitra, J. A. Morrison, and K. G. Ramakrishnan, "Virtual private networks: joint resource allocation and routing design," in *Proceedings of the 18th Annual Joint Conference of the IEEE Computer and Communications Societie (INFOCOM '99)*, pp. 480–490, March 1999.

[2] F. P. Kelly, "Kelly blocking probabilities in large circuit-switched networks," *Advances in Applied Probability*, vol. 18, no. 2, pp. 473–505, 1986.

[3] K. W. Ross, *Multiservice Loss Models for Broadband Telecommunication Networks*, Springer, New York, NY, USA, 1995.

[4] W. Whitt, "Blocking when service is required from several facilities simultaneously," *AT&T Technical Journal*, vol. 64, no. 8, pp. 1807–1856, 1985.

[5] H. Lian and A. Faragó, "A heuristic optimization method and its application in cognitive radio networks," Computer Science Technical Report UTDCS-39-11, University of Texas at Dallas, Dallas, Tex, USA, 2011.

[6] B. Dengiz, F. Altiparmak, and A. E. Smith, "Local search genetic algorithm for optimal design of reliable networks," *IEEE Transactions on Evolutionary Computation*, vol. 1, no. 3, pp. 179–188, 1997.

[7] O. C. Martin and S. W. Otto, "Combining simulated annealing with local search heuristics," *Annals of Operations Research*, vol. 63, pp. 57–75, 1996.

[8] Y. Leung, G. Li, and Z. B. Xu, "A genetic algorithm for the multiple destination routing problems," *IEEE Transactions on Evolutionary Computation*, vol. 2, no. 4, pp. 150–161, 1998.

[9] M. Pioro and D. Medhi, *Routing, Flow, and Capacity Design in Communication and Computer Networks*, Morgan Kaufmann, San Fransisco, Calif, USA, 2004.

[10] H. Lian and A. Faragó, "Optimizing Call Admission Control for Cognitive Radio Networks Using a New Heuristic Optimization Method," *Science Academy Transactions on Computer and Communication Networks (SATCCN)*, vol. 2, no. 1, pp. 2046–5157, 2012.

[11] R. B. Cooper and S. Katz, "Analysis of alternate routing networks account taken of the nonrandomness of overflow traffic," Tech. Rep., Bell Telephone Laboratories Memorandum, 1964.

[12] A. A. Jagers and E. A. Van Doorn, "On the continued Erlang loss function," *Operations Research Letters*, vol. 5, no. 1, pp. 43–46, 1986.

[13] F. P. Kelly, "Loss networks," *Annals of Applied Probability*, vol. 1, no. 3, pp. 319–378, 1991.

[14] S. P. Chung and K. W. Ross, "Reduced load approximations for multirate loss networks," *IEEE Transactions on Communications*, vol. 41, no. 8, pp. 1222–1231, 1993.

[15] J. Arabas and S. Kozdrowski, "Applying an evolutionary algorithm to telecommunication network design," *IEEE Transactions on Evolutionary Computation*, vol. 5, no. 4, pp. 309–322, 2001.