A Recursive Random Search Algorithm for Optimizing Network Protocol Parameters

Tao Ye, Shivkumar Kalyanaraman
Department of Electrical, Computer and System Engineering
Rensselaer Polytechnic Institute
Troy, New York 12180
yet3@networks.ecse.rpi.edu, shivkuma@ecse.rpi.edu

Abstract

The performance of several network protocols can be significantly enhanced by tuning their parameters. The optimization of network protocol parameters can be modeled as a "black-box" optimization problem with unknown, multi-modal and noisy objective functions. In this paper, a recursive random search algorithm is proposed to address this type of optimization problems. The new algorithm takes advantage of the favorable statistical properties of random sampling and achieves high efficiency without imposing extra restrictions, e.g., differentiability, on the objective function. It is also robust to noise in the evaluation of objective function since it use no traditional noise-susceptible local search techniques. The proposed algorithm is tested on classical benchmark functions and its performance compared with a multi-start hillclimbing algorithm. The algorithm is also integrated with a new on-line simulation system which attempts to automate network management by tuning protocol parameters when network conditions change significantly. We present the application of this on-line simulation system in enhancing the performance of network protocols, such as, RED and OSPF.

1 Introduction

Optimization problems arise in many engineering areas and can be formulated as (assume minimization): given a real-valued objective function $f : \mathbb{R}^n \to \mathbb{R}$, find a global minimum,

$$\mathbf{x}^* = \arg\min_{\mathbf{x}\in D} f(\mathbf{x}) \tag{1}$$

where D is called parameter space, usually a compact set in $\mathbb{R}^n$. The most common feature in these practical problems is that the objective function $f(\mathbf{x})$ is not known analytically and the function evaluation can only be achieved through computer simulation or other indirect ways. In other words, the objective function can be viewed as a black box which outputs the function value when given the parameters as input. The optimization of this type of objective functions are so-called "black-box" optimization problem. Essentially, the search algorithms for solving such problems involve sampling the parameter space D and evaluating $f(\mathbf{x})$ for each sample $\mathbf{x}$ until the global optimum is found. Since little *a priori* knowledge about the "black-box" is assumed, these problems are considered very hard to solve. Another feature making the problems even harder is that the objective functions are often non-linear and multi-modal, therefore they are also called *global optimization* as opposed to local optimization where there is only one single extreme in $f(\mathbf{x})$ and are much easier to solve with many powerful techniques available.

In the context of network optimization, the performance of network protocol sensitive to its parameter vector $\mathbf{x}$ can be modeled as $f(\mathbf{x})$. Here $f(\mathbf{x})$ can be evaluated for any $\mathbf{x}$ through a simulation. Based on this concept, an on-line simulation system has been proposed in [1], which attempts to accomplish automatic and adaptive network management by use of protocol parameter optimization. The basic idea is illustrated in Fig 1. The on-line simulator in a network domain continuously collects real-time data on

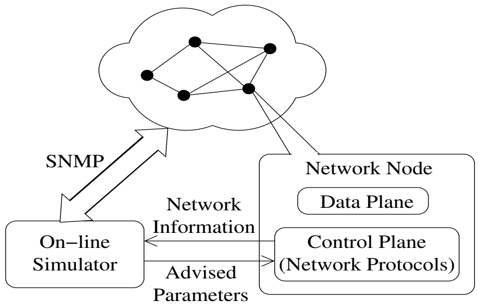

Figure 1: On-line simulation scheme for automatic network management

network conditions, and based upon triggering conditions, runs network simulations (i.e., evaluates $f(\mathbf{x})$) and uses a search algorithm to find better parameters for network protocols. A good search algorithm, which can *quickly* find better protocol parameters, is essential to the success of this on-line simulation system. In this paper, we will address the problem of designing such a search algorithm.

Besides the difficulties common in all "black-box" optimization problems, some specific properties in the network optimization context are:

High efficiency: Network conditions keep changing and hence the search algorithm must *quickly find better network parameters* before significant changes in the network happen. Accordingly, the emphasis of the algorithm should not be on seeking the strictly global optimum, but finding a better operating point within the limited time frame. The high efficiency requirement is also due to the fact that the simulation of a complex network is often very time-consuming and it is necessary to reduce the number of function evaluations as much as possible.

Scalability to high-dimensional problems: A network often has hundreds or thousands of protocol parameters to be tuned. Usually high-dimensional problems are much more difficult to solve than low-dimensional problems[2].

Robustness to noise: Network simulation only provides us with an approximate evaluation of $f(\mathbf{x})$. This means the objective functions in our problems are superimposed with small random noise due to inaccuracy in network modeling, simulation, etc..

In recent years, "black-box" optimization has become a very active research area and various search algorithms have been proposed and successfully applied in practice. Among them, stochastic algorithms have achieved wide popularity, which introduce stochastic elements into the search process and hence can only guarantee a probabilistic convergence to the global optimum. Their success is mainly because they impose few restrictions on the objective functions, moreover, due to their simplicity they are mostly easy to implement and adapt to complex problems. Pure random search is the simplest stochastic algorithm, which performs uniform random sampling in the parameter space. Although pure random search is considered to lack efficiency, many other search algorithms (e.g., multi-start type algorithms) include it as a part to provide a probabilistic guarantee of convergence to the global optimum. Controlled random search[3] uses simplex sampling technique to explore the parameter space. The idea behind simplex sampling is to first uniformly sample the parameter space and then increasingly bias the sampling towards the good regions found in the previous search. It has been demonstrated to be very effective in many practical problems. Multi-start hillclimbing tries to improve the efficiency of pure random search by starting a hillclimbing search process from each random sample until reaching a local optimum. However, it may waste a lot of time on re-climbing the hills already visited. Clustering method[4] attempts to avoid the revisit of the hills by identifying the random samples close to each other as a cluster and only starting one hillclimbing process from each cluster in the hope that a cluster only includes one local optimum. However, the problem is: it is very difficult to accurately identify these clusters especially in high dimension problems[4]. Simulated annealing[5] and Genetic algorithm[6] are two widely used techniques inspired by natural phenomenon, i.e., physical annealing process and natural evolution process. Although they have been successfully used in many practical problems, their efficiencies are usually low and the algorithms often take a long time to converge. Many variants have been attempted to combine them with some local search methods to improve efficiency.[7]

The No Free Lunch theorem[8] has demonstrated that no single algorithm can consistently perform better in every class of problems than the other algorithms. That is, for one class of problems where an algorithm can do better than other algorithms, there is always another class where it will perform poorly. For one specific optimization problem, the most efficient algorithms are those which best exploit the available information of the objective function. This paper examines the properties of network optimization problems and proposes a recursive random search algorithm which is geared to the requirements and restrictions of network optimization problems and attempts to achieve high efficiency with little *a priori* knowledge assumed.

The rest of this paper is organized as follows: in Section 2, we discuss design concepts of our recursive random search algorithm. In Section 3, we describe the details of the new algorithm. In Section 4, the algorithm is tested on classical benchmark functions, and its performance compared with other stochastic algorithms. In Section 5, we apply the algorithm to two real network optimization problems, RED and OSPF, and show how the performance of these network protocols can be improved. Finally, we conclude this paper in Section 6 and discuss further research directions.

2 Recursive Random Search (RRS): Design Ideas

A stochastic search algorithm consists of two procedures: *exploration* and *exploitation*, which are also referred to as *global phase* and *local phase* in some literature[2, 4]. Exploration encourages the search process to examine unknown regions, while exploitation attempts to converge to a local optimum in the vicinity of a chosen region. They can be executed alternately or be interwoven together during the search. The balance between these two procedures is maintained by a *balance strategy* which can be either probabilistic or deterministic. Their relationship is illustrated in Fig 2. For example, multi-start hillclimbing uses ran-

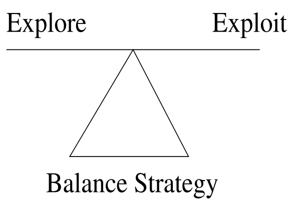

Figure 2: Exploration, Exploitation and Balance Strategy

dom sampling for exploration, a local search technique for exploitation and a deterministic balance strategy to execute the two procedures alternately. In the following, we will investigate the applicability of various exploration and exploitation techniques, choose those which best fit our problem, and then design an appropriate balance strategy to combine these methods to achieve high efficiency.

Exploration methods are usually required to provide a probabilistic guarantee of convergence to the global optimum. Not many techniques are available in this category. Some of the examples are deterministic enumeration, random sampling, random walk, etc. Among them, random sampling may be the simplest and most widely used technique, which takes random samples from a uniform distribution over the parameter space. This technique provides a strong probabilistic convergence guarantee and is more effective than other methods, especially for high-dimensional problems[4]. Therefore, we have chosen random sampling for exploration in our algorithm.

Random sampling is generally considered to lack efficiency, however, we will show in the following that it is in fact very efficient in identifying a good point close to promising areas in its initial steps. Suppose we have an objective function $f(\mathbf{x})$ defined on the parameter space D and the range of $f(\mathbf{x})$ is $[y_{min}, y_{max}]$, then given a number $y_r \in [y_{min}, y_{max}]$, we can define the following percentile function which stands for the portion of the points in the parameter space whose function values are smaller than y_r:

$$\phi_D(y_r) = \frac{m(\{\, \mathbf{x} \in D \mid f(\mathbf{x}) \leq y_r \,\})}{m(D)} \tag{2}$$

where $m(\cdot)$ is *Lebesgue measure*, a measure of the size of a set. For example, *Lebesgue measure* in a 2-dimensional space is just area, and volume in a 3-dimensional space, and so on. $\phi_D(y_r)$ is a monotonously increasing function with a range of $[r_{min}, 1]$. Here r_{min} is a value between 0 and 1. If $r_{min} > 0$, that means the size of the set of global optimum points is not zero, for example, in a 2-dimensional parameter space, this means the global optima will consist not of discrete points but some areas. Without loss of generality, we consider $r_{min} = 0$ in the following discussion. Therefore, we have $\phi_D(y_{min}) = 0$ and $\phi_D(y_{max}) = 1$. If we have a y_r such that $\phi_D(y_r) = r, \quad r \in [0, 1]$, then we can define a *r-percentile* set in the parameter space D:

$$A_D(r) = \{\, \mathbf{x} \in D \mid f(\mathbf{x}) \leq y_r \,\} \tag{3}$$

Note that $A_D(1)$ is just the whole parameter space D and $\lim_{\epsilon \to 0} A_D(\epsilon)$ will converge to the global optima. Suppose the sample sequence generated by n steps of random sampling is $\mathbf{x}_i, i = 1 \ldots n$ and $\mathbf{x}^n_{(1)}$ is the one with the minimum function value, then the probability of $\mathbf{x}^n_{(1)}$ in $A_D(r)$ is:

$$P(\mathbf{x}^n_{(1)} \in A_D(r)) = 1 - (1 - r)^n = p \tag{4}$$

Alternatively, we can compute the r value of the *r-percentile* region that $\mathbf{x}^n_{(1)}$ will reach with probability p as:

$$r = 1 - (1 - p)^{1/n} \tag{5}$$

Note that Equation 5 is just the inverse of 4. For any probability $p < 1$, r will tend to 0 with increasing n, that means, random sampling will converge to the global optima with increasing number of samples. From Equation 5, we can also see that *random sampling is highly efficient at initial steps since r decreases exponentially with increasing n, and its inefficiency is from later samples*. For example, Fig 3 shows the r-percentile region that random sampling can reach with 99% probability. We can see that it takes only 44 samples to reach a point in $A_D(0.1)$ area and this is achieved without using any special information of the objective function, whereas all samples after the first 44 can only improve r value of $\mathbf{x}^n_{(1)}$ at most by 0.1.

Exploitation methods are required to quickly converge to the local optimum in a certain region. The derivative-based methods, such as quasi-Newton method[9] and deepest descent[10], are not adopted in our algorithm though they are very efficient for differentiable objective functions. The reason is that the objective function in our problem may be superimposed with noise and hence the accurate derivative estimation cannot be obtained. Direct search methods, such as Nelder-Mead simplex method[11] and pattern search[12], do

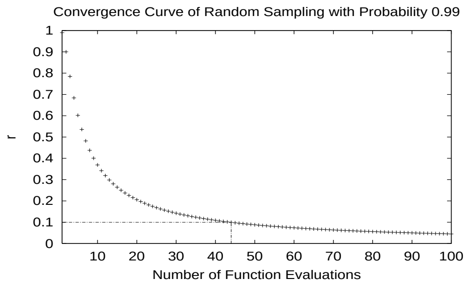

Figure 3: Convergence curve of random sampling with probability 0.99

not exploit the derivative of the objective functions, are often recommended for optimization in the presence of noise[13, 14]. However, like other local search algorithms, they are still susceptible to the effect of noise and their performance may greatly degrade in high-dimensional problems with many parameters[15, 4]. Therefore, we propose a new direct search approach in our algorithm which is based on the high efficiency of random sampling at initial steps. This approach starts a random sampling in the prospective region with a relatively large sample space, then *shrinks* and *re-aligns* the sample space based on previous samples until it finally converges to the local optimum. The details of this approach are described in the next section. Without noise-susceptible and poorly-scalable local search methods involved, our approach is more robust to noise in function evaluations and more scalable. These advantages are achieved without sacrificing the efficiency since the method takes advantage of the initial high-efficiency property of random sampling.

balance strategy is essential to the efficiency of an algorithm. Ideally it should only execute the exploitation procedure in prospective areas and explore otherwise. However, it is difficult to judge which areas are more promising and should be exploited. Many algorithms, such as multi-start type algorithms, do not differentiate between areas and hence waste much time in less promising areas. Our approach is to identify a certain *r-percentile* region $A_D(\gamma)$ and only start exploitation from the points in this region. The size of $A_D(\gamma)$ region should be small so that most of less promising areas are filtered out. On the other hand, it should be within the reach of initial highly efficient steps of random sampling so that identifying a point in it will not take too long to lower the overall efficiency. Fig 4 illustrates an example of this strategy. The left plot shows a contour plot of a 2-dimension multi-modal objective function and the right plot shows the region of $A_D(0.05)$. We can see the function has many local optima; however, only 3 discrete areas remain in $A_D(0.05)$ (shaded areas in the right plot). Each of these areas encloses a local optimum and the one with the biggest size happens to contain the global optimum. This is often true for many optimization problems since the region of attraction containing the global optimum usually has the largest size [2]. If we do random sampling on the whole parameter space, those samples contained in $A_D(\gamma)$ are still uniformly distributed over $A_D(\gamma)$, i.e., they are more likely to belong to the region containing the global optimum. Therefore, if we start exploitation from these points, the search will arrive at a global optimum with a larger probability than other non-global optima.

3 Recursive Random Search: Details

The basic idea of the RRS algorithm is to use random sampling to explore the whole parameter space and only start exploitation for those points which fall in a certain $A_D(\gamma)$ region. The pseudo-code of the algorithm is shown in Algorithm 1 and we will explain its details in the following with reference to the lines of the pseudo-code.

Algorithm 1: Recursive Random Search Algorithm

Initialize exploration parameters $p, r, n \leftarrow \ln(1-p)/\ln(1-r)$;
Initialize exploitation parameters $q, v, c, s_t, l \leftarrow \ln(1-q)/\ln(1-v)$;
Take n random samples $x_i, i = 1 \ldots n$ from parameter space D;
$\mathbf{x}_0 \leftarrow \arg\min_{1 \leq i \leq n}(f(\mathbf{x}_i))$, $f_\gamma \leftarrow f(\mathbf{x}_0)$, add $f(\mathbf{x}_0)$ to the threshold set $\mathbf{F}$;
$i \leftarrow 0$, $flag \leftarrow 1$, $\mathbf{x}_{opt} \leftarrow \mathbf{x}_0$;
while *stopping criterion is not satisfied* **do**
    if $flag = 1$ **then**
        // Exploit flag is set, start exploitation process
        $j \leftarrow 0$, $f_l \leftarrow f(\mathbf{x}_0)$, $\mathbf{x}_l \leftarrow \mathbf{x}_0$, $\rho \leftarrow r$;
        while $\rho > s_t$ **do**
            Take a random sample $\mathbf{x}'$ from $N_{S,\rho}(x_l)$;
            if $f(\mathbf{x}') < f_l$ **then**
                // Find a better point, re-algin the center of sample space to the new point
                $\mathbf{x}_l \leftarrow \mathbf{x}'$, $f_l \leftarrow f(\mathbf{x}')$;
                $j \leftarrow 0$;
            else
                $j \leftarrow j + 1$;
            endif
            if $j = l$ **then**
                // Fail to find a better point, shrink the sample space
                $\rho \leftarrow c \cdot \rho$, $j \leftarrow 0$;
            endif
        endw
        $flag \leftarrow 0$, update $\mathbf{x}_{opt}$ if $f(\mathbf{x}_l) < f(\mathbf{x}_{opt})$;
    endif
    Take a random sample $\mathbf{x}_0$ from S;
    if $f(\mathbf{x}_0) < f_\gamma$ **then**
        // Find a promising point, set the flag to exploit
        $flag \leftarrow 1$;
    endif
    if $i = n$ **then**
        // Update the exploitation threshold every n samples in the parameter space
        Add $\min_{1 \leq i \leq n}(f(\mathbf{x}_i))$ to the threshold set $\mathbf{F}$;
        $f_\gamma \leftarrow \text{mean}(\mathbf{F})$, $i \leftarrow 0$;
    endif
    $i \leftarrow i + 1$;
endw

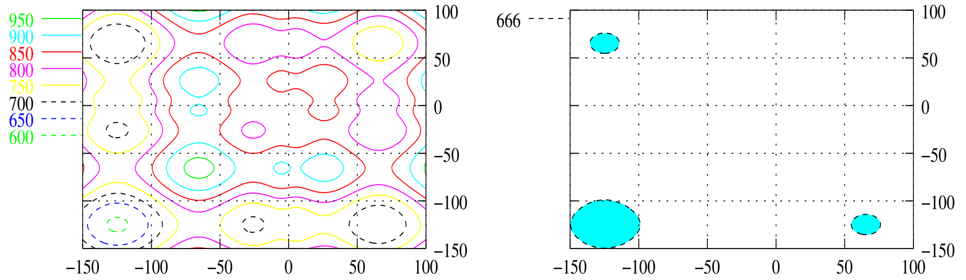

Figure 4: Contour plot of an objective function(left) and its region of $A_D(0.05)$(right)

Exploration phase: In this phase, we try to identify an $A_D(\gamma)$ area for exploitation. We first decide the threshold value y_γ for $A_D(\gamma)$ such that any point with a smaller function value than y_γ belongs to $A_D(\gamma)$. Our algorithm uses the following strategy to decide the value of y_γ: first we choose a relatively small value $r \in [0, 1]$, and a confidence probability p, e.g., 0.99, then we take n random samples where n is the smallest integer so that the probability of the best point $\mathbf{x}^n_{(1)} \in A_D(r)$ is larger than p. Consider Equation 4: if we write n in terms of other variables, we have $n = \frac{\ln(1-p)}{\ln(1-r)}$ (line 1 in pseudo-code). We use $f(\mathbf{x}^n_{(1)})$ as the initial value of y_γ and add it into the threshold set $\mathbf{F}$ (line 4). In later exploration, we obtain a new $\mathbf{x}^n_{(1)}$ every n samples and update y_γ with the average of the set $\mathbf{F}$ (lines [21-23]). In this method, the confidence probability p should assume a value close to 1, for example, 0.99. The value of r decides the value of γ (γ is smaller than r since y_γ is the function value of some point in $A_D(r)$) and hence the size of $A_D(\gamma)$. It should be chosen carefully to balance efficiency and effectiveness as discussed before. We have used $r = 0.1$ and $p = 0.99$ in our current algorithm, and in this case it only takes 44 samples to find a point which falls within $A_D(0.1)$.

Exploitation phase: As soon as *exploration* finds a promising point $\mathbf{x_0}$ with a function value smaller than y_γ, we start *exploitation*, a recursive random sampling procedure, in the neighborhood $N(\mathbf{x}_0)$ of $\mathbf{x_0}$. The size of $N(\mathbf{x}_0)$ is taken as $r \cdot m(D)$, where D is the original parameter space and r the parameter in exploration process. Since our exploration process guarantees that with a high probability $\mathbf{x}_0$ belongs to $A(\gamma)$ whose size is smaller than $r \cdot m(D)$, the above size will ensure $N(\mathbf{x}_0)$ covers a local optimum with a high probability. Currently a simple method is used to construct $N(\mathbf{x}_0)$: Assume a n-dimension parameter space S defined with $l_i \leq x_i \leq u_i, i = 1 \ldots n$, where x_i is ith element of a point $\mathbf{x} \in S$ and u_i, l_i are its upper and lower limits. The neighborhood of $\mathbf{x}_0$ with a size of $r \cdot m(S)$ (line 10) is: $N_{S,r}(\mathbf{x_0}) = \{z \in S \mid |z_i - x_{0,i}| < r^{1/n} \cdot (u_i - l_i)\}$, where $x_{0,i}$ is ith element of $\mathbf{x}_0$ and z_i ith element of $\mathbf{z}$. During exploitation phase, two sub-phases, i.e., re-align and shrink, are alternately performed until the search converges to the local optimum.

Re-align sub-phase: As described above, we first start exploitation in the neighborhood $N(\mathbf{x}_0)$ of $\mathbf{x}_0$. If $\phi_{N(\mathbf{x}_0)}(f(\mathbf{x}_0))$ (defined in Equation 5) is large, that means most points in $N(\mathbf{x}_0)$ are better than $\mathbf{x}_0$. Therefore, if we do random sampling in $N(\mathbf{x}_0)$, it will be highly likely to find a point better than $\mathbf{x}_0$ with a small number of samples. Assuming $\phi_{N(\mathbf{x}_0)}(f(\mathbf{x}_0)) = \upsilon$, with a confidence probability q, random sampling should find a better point in $N(\mathbf{x}_0)$ with $l = \frac{\ln(1-q)}{\ln(1-\upsilon)}$ (line 2) samples. If a better point is found in l samples, we replace $\mathbf{x}_0$ with this point, move the sample space to the new $N(\mathbf{x}_0)$ and keep its size unchanged (lines [11-13]). This is called a "re-align" operation For example, Fig 5 illustrates the behavior of RRS in a 2-dimensional search space. We first obtain a promising point C_1 from the exploration process and then start exploitation (i.e., random sampling) in the neighborhood R_1 of C_1. After a few samples, we find a point C_2 in R_1 better than C_1 and then move the sample space from R_1 to the neighborhood R_2 of C_2. In this way, even if $N(\mathbf{x}_0)$ (i.e., R_1 in the example) might not include the local optimum, the consecutive re-align moves

will still lead the search to converge to the local optimum.
Shrink sub-phase: If we fails to find a better point in l samples, that suggests $\phi_{N(\mathbf{x}_0)}(f(\mathbf{x}_0))$ is small. In this case, we reduce $N(\mathbf{x}_0)$ by a certain ratio $c \in [0, 1]$, i.e., generate a new neighborhood $N'(\mathbf{x}_0)$ whose size is $c \cdot m(N(\mathbf{x}_0))$ (lines [15-16]). For example, as shown in Fig 5, after l unsuccessful samples in R_2, we shrink the sample space to R_3, then to R_4 if sampling in R_3 continue to fail. Note again that this shrink sub-phase is done only when we *fail* to find a better point in l samples. With this method, the search will soon converge to the local optimum if it is included in the current sample space, or to an appropriate size of sample space where $\phi_{N(\mathbf{x}_0)}(f(\mathbf{x}_0))$ is v, then moves to the local optimum by the re-align procedure described above. The whole exploitation process continues until $m(N(\mathbf{x}_0))$ (Recall $m(\cdot)$ is the *Lebesgue* measure of a set) is less than a certain threshold, whose value is dependent on the resolution requirement of the optimization problem.

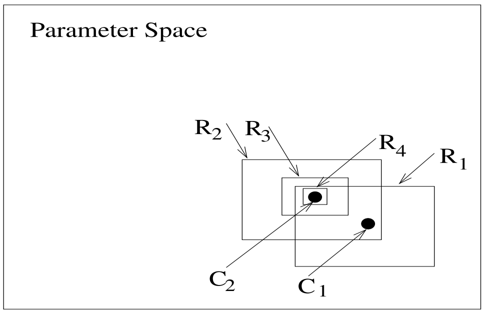

Figure 5: Shrink and Re-align Process

4 Tests on Standard Benchmark Functions

As we described before, the objective in the design of Recursive Random Search is: high efficiency, scalability to high-dimensional problems and robustness to function evaluation noise. In the following, we will test the performance of RRS in these three aspects.

4.1 Benchmark Functions

We have used a suite of classical benchmark functions in our performance tests. These functions are used in many literature for testing purpose and considered difficult to optimize. We describe these function in the following. For these functions, we assume that the parameter $\mathbf{x}$ is a n-dimensional vector and x_i denotes its ith element.

- Schwefel's (Sine Root) Function[2] is defined as:

$$f(\mathbf{x}) = V \cdot n + \sum_{i=1}^{n} (-x_i \cdot \sin(\sqrt{|x_i|}) \tag{6}$$

  where $-500 \le x_i \le 500, i = 1 \ldots n$ and $V = 418.9829$. The global minimum is 0 at $x_i = 420.9687, i = 1 \ldots n$. There is a second-best minimum at $x_i = 420.9687$, $i = 1 \ldots n, i \neq j$, $x_j = -302.5232$. This second-best minimum is a long way from the global minimum and some algorithms can be trapped in the wrong region. A 2-dimensional Schwefel function is shown in Fig 6.

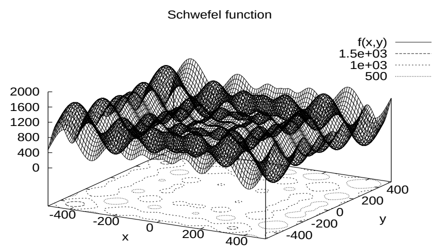

Figure 6: Schwefel's Function

- Rastrigin's Function was proposed by Rastrigin[16] and generalized as the following format by Mühelenbein, Schomisch and Born[17]

$$f(\mathbf{x}) = n \cdot A + \sum_{i=1}^{n} (x_i^2 - A \cdot \cos 2\pi x_i) \tag{7}$$

where $-5.12 \leq x_i \leq 5.12$ and $A = 10$. The global minimum is 0 at $x_i = 0$. There are a large number of local minima in this function, for example, grid points with $x_i = 0$ except one coordinate, where $x_j = 1.0$, give $F(x) = 1.0$. It is considered difficult for most methods. A 2-dimensional Schwefel function is shown in Fig 7.

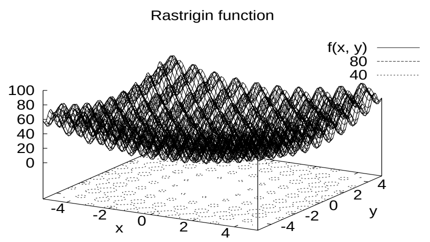

Figure 7: Rastrigin Function

- Rosenbrock's Saddle[18] is defined as:

$$f(x) = \sum_{i=1}^{n-1} (100 \cdot (x_{i+1} - x_i)^2 + (1 - x_i)^2) \tag{8}$$

where $-2.048 \leq x_i \leq 2.048$. The global minimum is 0 at $x_i = 1.0$. This function has a long curved valley which is only slightly decreasing. There are strong interactions between variables. A 2-dimensional Schwefel function is shown in Fig 8.

- Griewangk's Function is one of the most difficult global optimization test functions[2]:

$$f(\mathbf{x}) = 1 + \sum_{i=1}^{n} (x_i^2/4000) - \prod_{i=1}^{n} (\cos(x_i/\sqrt{i})) \tag{9}$$

where $-600 \leq x_i \leq 600$. The global minimum is 0 at $x_i = 0$. Typical n is 10, for which there are four local minima $f(x) \approx 0.0074$ at $x \approx (\pm\pi, \pm\pi \cdot \sqrt{2}, 0, ..., 0)$. This function has a product

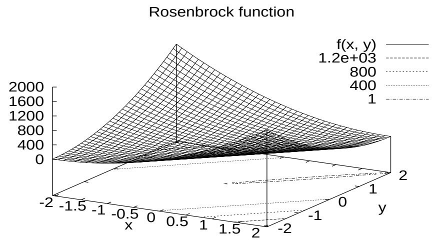

Figure 8: Rosenbrock's Saddle Function

term, introducing an interdenpendency between the variables. This is intended to disrupt optimization technqiues that work on one function variable at a time. A 2-dimensional Griewangk function is shown in Fig 9.

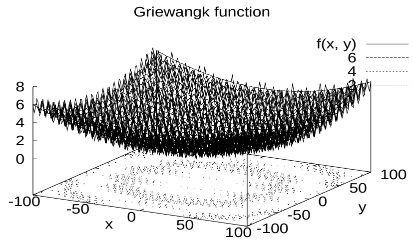

Figure 9: Griewangk's Function

A modified Griewan function varies the weight of the quadratic term:

$$f_\sigma(\mathbf{x}) = 1 + \sigma \sum_{i=1}^{n} (x_i^2/4000) - \prod_{i=1}^{n} (\cos(x_i/\sqrt{i})) \tag{10}$$

This function is a bumpy quadratic when σ is one and a product of cosines when σ is zero. The value of σ decides the difference between the local minima. When σ approaches 0, the values of local minima become similar.

- Ackley's Function is defined as:

$$f(x) = 20 + e - 20 \cdot e^{-0.2 \cdot \sqrt{\frac{1}{n} \cdot \sum_{i=1}^{n} x_i^2}} - e^{\frac{1}{n} \cdot \sum_{i=1}^{n} \cos 2\pi x_i} \tag{11}$$

where $-30 \leq x_i \leq 30$. The global minimum is 0 at $x_i = 0$ A 2-dimensional Ackley function is shown in Fig 10.

4.2 Tests on Efficiency of RRS

We have tested the RRS algorithm on the benchmark functions described before and compared its efficiency with one other heuristic search algorithm, multi-start hillclimbing. We have chosen multi-start hillclimbing for comparison because it has been demonstrated to work very well for many practical problems, for example, it produced excellent solutions on practical computer vision tasks[19], outperformed simulated

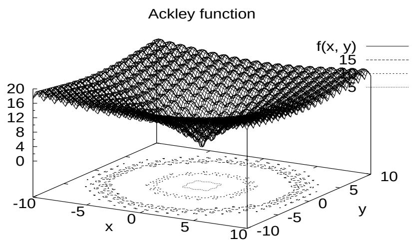

Figure 10: Ackley's Function

annealing on the traveling salesman problem (TSP)[20], and outperformed genetic algorithms and genetic programming on several large-scale testbeds[21]. Our version of multi-start hillclimbing uses as its local search method *pattern search*[12], one of direct search techniques which are usually recommended for black-box optimization problems[14]. To test the scalability of our algorithm, we apply the search algorithms to each function with its dimension n varying from 20 to 200. For eliminating random errors brough about by the randomness in the algorithms and the choice of starting points, we repeat each test with random starting points for 50 times and take the average of the testing results.

In the tests, we have chosen $A = 10$ for Rastrigin function and used the following parameters for our algorithm: $p = 0.99, r = 0.1, c = 0.5, q = 0.99, s_t = 0.01$. The testing results for each benchmark function are shown in Fig 11-16. We can see that our algorithm converges more rapidly and its efficiency is much better than multi-start hillclimbing for all test cases.

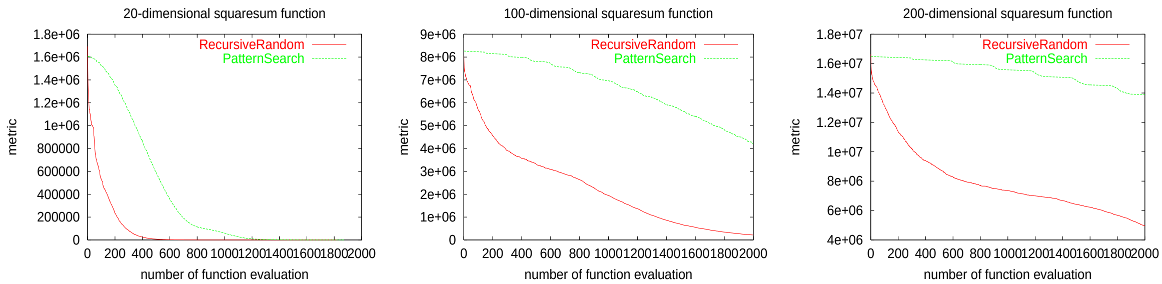

Figure 11: Tests on SquareSum Function

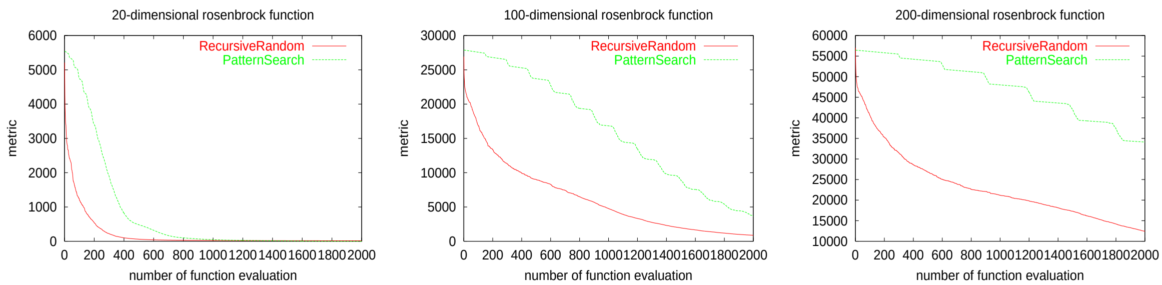

Figure 12: Tests on Rosenbrock Function

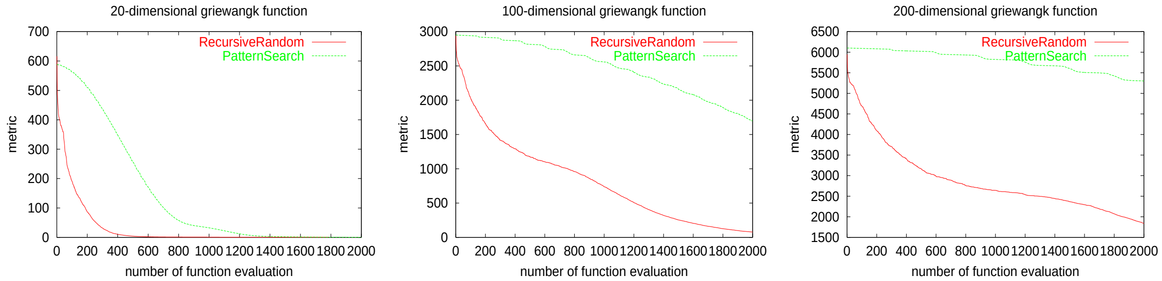

Figure 13: Tests on Griewangk Function

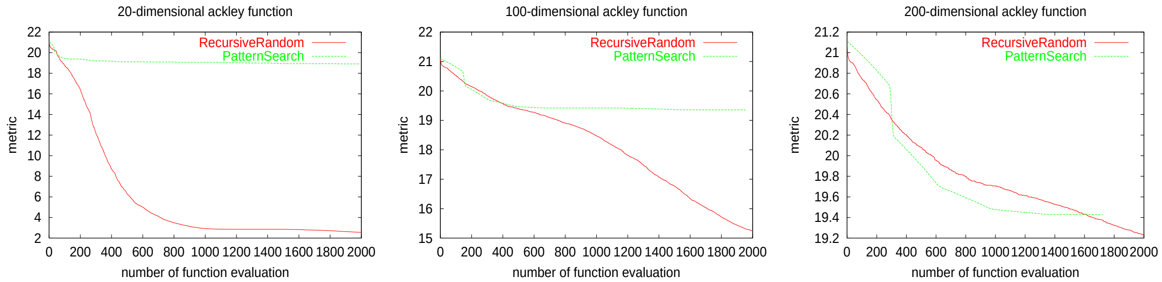

Figure 14: Tests on Ackley Function

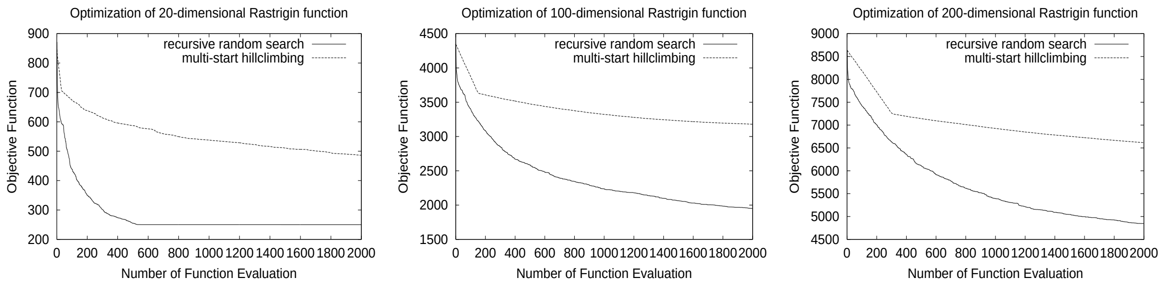

Figure 15: Tests on Rastrigin Function

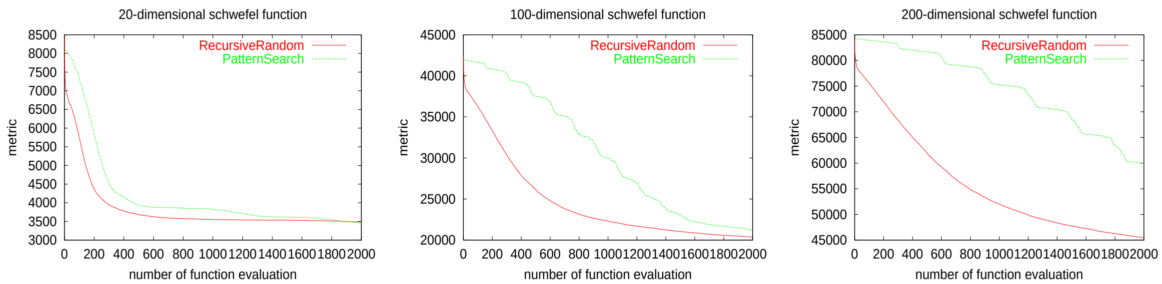

Figure 16: Tests on Schwefel Function

4.3 Tests on Robustness of RRS

We have used Rastrigion Function to emulate the situation where there is noise in the objective function. Recall that Rastrigin function is defined as follows:

$$f(\mathbf{x}) = n \cdot A + \sum_{i=1}^{n} (x_i^2 - A \cdot \cos(2\pi x_i)) \tag{12}$$

This function can be considered as a simple sphere function $\sum_{i=1}^{n} x_i^2$ superimposed with noise $\sum_{i=1}^{n} A \cdot \cos(2\pi x_i)$. The level of noise is decided by the value of A. To test the robustness of our algorithm to noise, we vary the noise level of Rastrigin function, i.e., the value of A and see how the performances of the search algorithms are affected. Note that the noise level should not be too large to totally distort the original function to another one. As shown in the Figure 17, with increased noise level, the performance of multi-start hillclimbing greatly degrades while our algorithm is much more robust to the effect of noise.

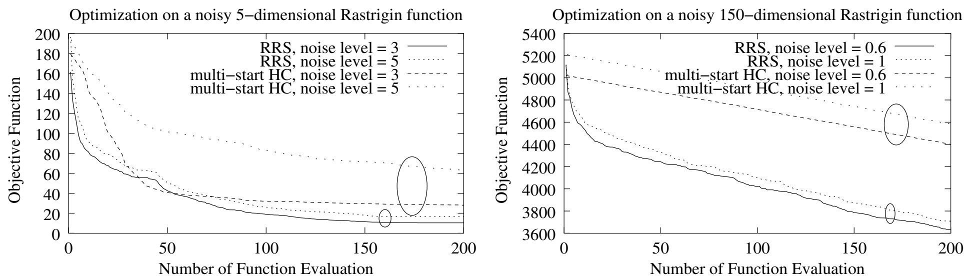

Figure 17: Robustness test of search algorithms

5 Application in Network Optimization

Recall that our goal in designing RRS is to model network parameter tuning as a black-box optimization problem and integrate RRS with the on-line simulation system (as shown in Fig 1) to perform dynamic network parameter optimization. Note again that on-line simulation is used in this context to evaluate $f(\mathbf{x})$ for each choice of $\mathbf{x}$. We have applied this on-line simulation system with RRS to selected network problems and here we present these applications.

5.1 Optimize RED Performance

The first application we tested is a Random Early Detection (RED) queue optimization problem. A RED queue has 4 parameters: *min threshold*, *max threshold*, *max drop probability* and *exponential weighted moving average coefficient*, and it has been known to be difficult to tune these parameters to different network conditions[22]. Now we will apply our algorithm to tackle this problem.

In this test, we use a linux-based testbed with a network topology as shown in Fig 18. There are 4 routers, each with a RED queue on it. Thus we have a total of 16 parameters for these 4 RED queues. We generate TCP flows from one side to the other and use as the performance metric the coefficient of variation ($\frac{\sigma}{\mu}$) of goodputs for these TCP flows, which measures the variability of the goodputs. The on-line simulator previously described is applied in the network to tune the parameters of these RED queues. This on-line simulator uses *ns*[23] to simulate the network and our RRS algorithm to search for better RED parameter settings. If a parameter setting is found with a performance metric better than the current by a certain margin, the setting is applied into the network. The COV of goodputs during the experiment is plotted in Fig 19. In the beginning, the parameters of these RED queue are purposely set to bad values, which result in an unfairness, i.e., a high average COV value and big oscillations. At 325 second, the on-line simulator is started (as shown in Fig 19). Soon the appropriate parameter setting is found by our RRS algorithm and the RED parameters are reconfigured. This results in an immediate performance improvement shown in the plot: the average COV drops to a very low value and the instantaneous COV curve becomes very stable over time.

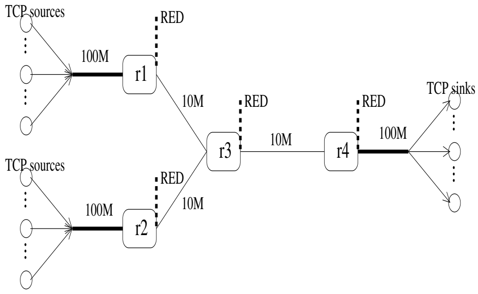

Figure 18: A 4-router testbed configuration

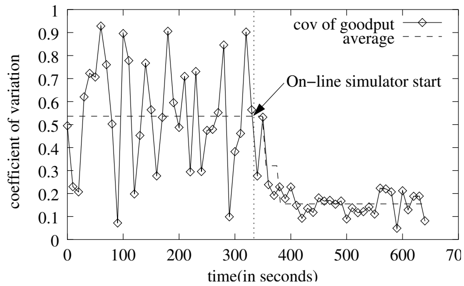

Figure 19: Optimize coefficient of variation of goodputs for TCP flows

5.2 Traffic Engineering by Tuning OSPF

Another application of our search algorithm is to perform traffic engineering by tuning link weights of OSPF algorithms. Given the demand matrix for a network, which defines the traffic flows between each source-destination pair, engineering is defined as the task of *appropriately* mapping these traffic flows onto an existing physical topology. One approach for performing traffic engineering is to set the link weights of OSPF with a knowledge of the demand matrix such that the resulting set of traffic flows is optimal in terms of some network-wide performance metric. We have chosen the average packet drop rate as the performance metric to optimize since packet drops have significant effect on the performance of underlying application and meanwhile total packet drops in a network are also an indicator of load balancing achieved. Suppose we are given a demand matrix, we can map the traffic flows onto the links in the network according to current OSPF routing. Thus we can calculate the average load on each link. Modeling link k as an GI/M/1/k queue, we can derive the drop probability P_k on this link in terms of it average load. Then we can obtain the networkwide average drop rate Φ by using the following formula:

$$\Phi = \sum_{k \in \mathcal{L}} \lambda_k P_k \tag{13}$$

where $\mathcal{L}$ is the set of links in the network, λ_k the average load on link k and P_k the drop probability on it. The problem of setting OSPF parameters (link weights) such that Φ is minimized is NP-hard[24]. Therefore, a search algorithm, such as the RRS algorithm, has to be used to find an approximate solution. We tested our RRS algorithm on 3 different networks with 80-140 links (i.e., 80-140 parameters). The results show that RRS is 50-90% faster than the local search method proposed in [25] and 30-60% fewer packet drops can be achieved by tuning OSPF link weights. The details of this work are presented in [24].

6 Conclusion and Future Work

In this paper, we investigate the optimization problem of network protocol parameters, which can be modeled as a "black-box" optimization problem. Any search algorithm intended to address this problem must be highly efficient, robust to noise and scalable to high-dimensional problems. Classical optimization algorithms, such as controlled random sampling, clustering method and simulated annealing, do not fit well in this context. A recursive random search (RRS) algorithm is proposed in this paper to address the problem. The algorithm is mainly based on random sampling and achieves high efficiency by exploiting the advantageous statistical property of random sampling. The RRS algorithm is tested on a suite of benchmark functions for efficiency, scalability and noise tolerance. We then integrate the algorithm into an on-line simulation system and apply it to real network optimization problems, such as RED and OSPF. In our tests, the performance of the protocols are greatly enhanced with the automatic tuning capability added by the on-line simulation system. Note that a non-trivial step in the application of RRS is to choose the correct subset of parameters (i.e., $\mathbf{x}$) and the optimization function (i.e., $f(\mathbf{x})$) with its constraints. In other words, posing parameter-tuning problem carefully to apply RRS is very important.

Besides the networking applications presented in this paper, we are examining other possible applications. For example, we are currently working on performing load balancing for outbound traffic by tuning BGP *Local_Pref* setting, which is a NP-hard problem. We are also investigating what is the class of NP-hard problems for which RRS can be an approximate solution. Many aspects of the RRS algorithm can also be further enhanced to achieve higher efficiency. For example, tabu technique can be included to reduce the probability of revisits. The parallelization of the algorithm can be another research direction to provide stronger ability to handle large-scale optimization problems.

References

[1] Traffic management and network control using collaborative on-line simulation. 2001.

[2] Aimo Törn and Antanas Žilinskas. *Global Optimization*, volume 350 of *Lecture Notes in Computer Science*. Springer-Verlag, 1989.

[3] W. L. Price. A controlled random search procedure for global optimization. In L. C. W. Dixon and G. P. Szegö, editors, *Towards Global Optimization 2*, pages 71–84. North-Holland, Amsterdam, Holland, 1978.

[4] A. H. Kan and G. T. Timmer. Stochastic global optimization methods part I: Clustering methods. *Mathematical Programming*, 39:27–78, 1987.

[5] S. Kirkpatrick, D.C. Gelatt, and M.P. Vechhi. Optimization by simulated annealing. *Science*, 220:671–680, 1983.

[6] Melanie Mitchell. *An Introduction to Genetic Algorithms*. The MIT Press, 1996.

[7] W. E. Hart. *Adaptive Global Optimization with Local Search*. PhD thesis, University of California, San Diego, 1994.

[8] D. h. Wolpert and W. G. Macready. No free lunch theorems for optimization. *IEEE Transaction on Evolutionary Computing*, 1:67–82, 1997.

[9] W. C. Davidon. Variable metric method for minimization. *SIAM Journal on Optimization*, 1:1–17, 1991.

[10] L. Armijo. Minimization of functions having lipschitz continuous first partial derivatives. *Pacific Journal of Mathematics*, 16:1–3, 1966.

[11] R. MEAD and J. A. NELDER. A simplex method for function minimization. *Computer Journal*, 7(4):308–313, 1965.

[12] R. Hooke and T. Jeeves. Direct search solution of numerical and statistical problems. *J.Assoc. Comp. Mach.*, 8(2):212–229, April 1961.

[13] P. Brachetti, M. De Felice Ciccoli, G. Di Pillo, and S. Lucidi. A new version of the price's algorithm for global optimization. *Journal of Global Optimization*, 10:165–184, 1997.

[14] Michael W. Trosset. On the use of direct search methods for stochastic optimization. Technical report, Department of Computational and Applied Mathematics, Rice University, 2000.

[15] Soraya Rana, L. Darrell Whitley, and Ronald Cogswell. Searching in the presence of noise. In H. Voigt, W. Ebeling, I. Rechenberg, and Hans-Paul Schwefel, editors, *Parallel Problem Solving from Nature – PPSN IV (Berlin, 1996) (Lecture Notes in Computer Science 1141)*, pages 198–207, Berlin, 1996. Springer.

[16] L. A. Rastrigin. *Systems of Extremal Control*. Nauka, 1974.

[17] H. Mühlenbein M. Schomisch and J. Born. The parallel genetic algorithm as function optimizer. In Richard K. Belew and Lashon B. Booker, editors, *Proceedings of the Fourth Intl. Conf. on Genetic Algorithms*, pages 271–278. Morgan-Kaufman, 1991.

[18] M. A. Wolfe. *Numerical Methods for Unconstrained Optimization*. Van Nostrand Reinhold Company, New York, 1978.

[19] J. Beveridge, C. Graves, and C. E. Lesher. Local search as a tool for horizon line matching. Technical Report CS-96-109, Colorado State University, 1996.

[20] D. S. Johnson and L. A. McGeoch. The travelling salesman problem: a case study in local optimizaiton. In E. H. L. Aarts and J. K. Lenstra, editors, *Local Search in Combinatorial Optimization*. Wiley and Sons, 1997.

[21] A. Juels and M. Wattenberg. Stochastic hillclimbing as a baseline method for evaluating generic algorithms. In D. S. Touretzky, M. C. Mozer, and M. E. Hasselmo, editors, *Advances in Neural Information Processing Systems*, volume 8, pages 430–436. 1996.

[22] S. Floyd and V. Jacobson. Random early detection gateways for congetsion avoidance. *IEEE/ACM Transactions on Networking*, 1:397–413, August 1993.

[23] NS. network simulator. http://www-mash.cs.berkeley.edu/ns.

[24] Dynamic optimization of ospf weights using online simulation. submitted.

[25] Bernard Fortz and Mikkel Thorup. Internet traffic engineering by optimizing ospf weights. In *Proceedings of the INFOCOM 2000*, pages 519–528, 2000.