A Recursive Random Search Algorithm for Large Scale Network Parameter Optimization

Abstract

Parameter configuration is a common procedure used in large scale network protocols to support multiple operational goals. This problem can be formulated as a black-box optimization problem and solved with on-line parameter optimization. This paper proposes a new heuristic search algorithm, Recursive Random Search(RRS), for large scale network parameter optimization. The RRS algorithm is based on the initial high-efficiency property of random sampling and attempts to maintain this high-efficiency by constantly "restarting" random sampling with adjusted sample spaces. Due to its root in random sampling, the RRS algorithm is robust to the effect of random noises in the objective function introduced by inaccuracies in network simulation. The RRS algorithm is also advantageous in optimizing the objective function with negligible parameters. The test results on the standard benchmark functions have demonstrated these properties. The RRS algorithm has been applied to the adaptive configuration optimization of some network protocols. One example application in tuning OSPF routing algorithm is presented.

1 Introduction

Today's network protocols like BGP and OSPF were designed for one primary service: "best effort reachability." But now operators want to deploy VPNs, manage traffic within ASes to meet SLA guarantees, and between ASes (at peering points) to optimize on complex peering agreements. The designers of such protocols had the insight to include "parametric hooks" to allow operators to "tweak" the protocols and achieve such traffic management goals. However, the parameter setting process today is manual and is widely considered a black art. The configuration of many protocols, such as BGP, is tough, error prone and is likely to get harder as the protocol is overloaded to server more functions[1]. Though some tools are emerging to aid operators, a lot more needs to be done. The on-line simulation

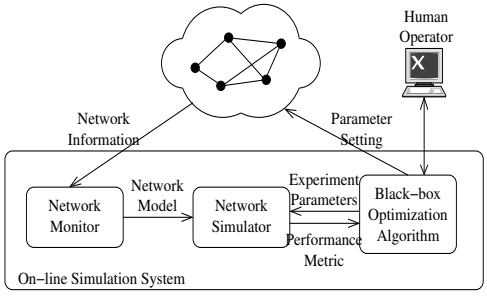

Figure 1: On-line simulation system for adaptive configuration of network protocols

system proposed in[2] is one such contribution to this important space. As illustrated in Fig 1, the basic idea of this system is to formulate network protocol configuration as a black-box optimization problem and perform the on-line simulation-based parameter optimization. This on-line protocol parameter tuning framework can dynamically adapt the protocol configuration to the prevailing network conditions whenever it detects major changes in the network. Various performance objectives can be achieved by performing optimizations for different performance metrics, such

as maximum network throughput or minimum end-to-end delay. This framework is also very flexible in that it can be easily used to manage any network protocol as long as it has tunable parameters and their setting has substantial effect on network performance. It is possible to be quickly deployed without incurring many changes on the existing network infrastructure. In addition, it may also be used as a "recommendation service" to suggest a variety of "good" parameter settings and illustrate resulting flow patterns so that operators are better informed than their current manual procedures.

In the on-line protocol tuning framework described above, an efficient search algorithm is essential to its success. The desired search algorithm should *quickly* find good protocol parameters within the time scale of variation in network conditions. Although a large number of optimization algorithms have been proposed and successfully applied in practice[3, 4, 5], there is no single algorithm which can consistently outperform the others in every class of problem[6]. For a specific type of problems, the performance of a search algorithm is dependent on whether its search strategy fit the features and requirements of the underlying problem. For network parameter optimization, the desired search algorithm should have the properties of high efficiency, scalability, scalability and noise-resistance, which traditional search algorithms, such as genetic algorithm and simulated annealing have been found insufficient to satisfy. This paper proposes a new optimization algorithm, Recursive Random Search(RRS), whose major feature is its basis on random sampling. The Recursive Random Search algorithm exploits the initial high efficiency of random sampling and remains in this high-efficiency phase by constantly restarting random sampling with adjusted sampling spaces. In addition, the RRS algorithm is also robust to noises in the objective function and is of great advantage when dealing with the objective function with negligible parameters.

The RRS algorithm has been tested on a suite of benchmark functions to examine its efficiency, noise-resistance and negligible parameter handling. The results show that RRS outperforms the other algorithms in comparison. We have successfully applied the RRS algorithm to the on-line tuning of several important network protocols, such as, RED, OSPF and BGP. And simulation results demonstrate substantial improvement in network performance when the proposed on-line tuning is deployed.

The rest of this paper is organized as follows: In Section 2, we examine features of network parameter optimization problems. In Section 3, we discuss design concepts of the RRS algorithm. In Section 4, we describe the details of this algorithm. In Section 5, the algorithm is tested on classical benchmark functions, and its performance compared with other popular stochastic algorithms. In Section 6, we apply the algorithm to one real network optimization problem, the tuning of OSPF routing algorithm. Finally, we conclude this paper in Section 7.

2 Network Parameter Optimization Problem

Like optimization problems arising in many engineering areas, network parameter optimization can be formulated as (assume minimization): given a real-valued objective function $f : \mathbb{R}^n \rightarrow \mathbb{R}$, find a global minimum,

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

where D is the predefined parameter space, usually a compact set in $\mathbb{R}^n$. In these problems, the objective function $f(\mathbf{x})$ is often analytically unknown and the function evaluation can only be achieved through computer simulation or other indirect ways. This type of problems are also called "black-box" optimization problems where the objective function is modeled as a black-box. Since little *a priori* knowledge about the black-box is assumed, these problems are considered very hard to solve. In addition, the objective functions are often non-linear and multi-modal, and these problems 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.

As mentioned before, designing or selecting an efficient search algorithm requires examining the features of the target problem. For network parameter optimization problems, the following features are usually present.

High efficiency is required for the desired search algorithm. More specifically, the emphasis of the search algorithm should be on finding a better operating point within the limited time frame instead of seeking the strictly global

optimum. Network conditions vary with time and the search algorithm should *quickly find better network parameters* before significant changes in the network occur. Furthermore, network parameter optimization is based on network simulation which is often very time-consuming. This also requires a highly efficient search algorithm which can obtain a desired solution with a minimum number of network simulations.

High dimension is another feature of these problems. For example, AT&T's network has 1000s of routers and links. If OSPF link weights of this network are to be tuned, there will be thousands of parameters present in the optimization. High-dimensional optimization problems are usually much more difficult to solve than low-dimensional problems because of "curse of dimensionality"[7].

Noises are often introduced into the evaluation of objective function since network simulation is used for function evaluations. Due to inaccuracies in network modeling, simulation, etc., this empirical evaluation of objective function may be distorted from the original one, in other words, affected by small random noises. Fig 2 shows an example of 2-dimensional empirical objective function obtained with network simulation. It can be seen that there exist many irregular small random fluctuations imposed on the overall structure.

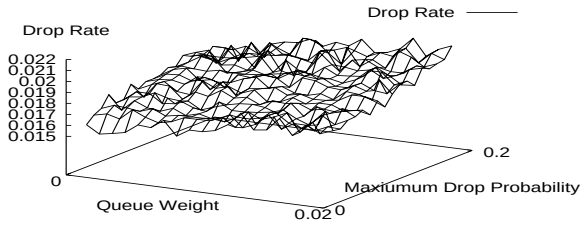

Figure 2: An empirical objective function obtained with network simulation

Negligible parameters may also be included in the objective function. These parameters contribute little to the objective function and should be ruled out from the optimization process. However, in practice, they are normally very difficult to be identified and eliminated effectively. If the search algorithm is able to automatically excluded these parameters from the optimization process, the efficiency of the optimization will significantly improved.

"Globally convex" or "big valley" structure[8, 9] may be present in the objective functions. This structure means that high-quality local optima tend to center around the global one with small average distance, whereas low-quality local optima tend to distribute around with larger average distance. "Globally convex" structure appears in many practical optimization problems, especially in the situations when the objective function is affected by random noises. Boese[10] has demonstrated the existence of this structure in complex Traveling Salesman Problem(TSP) and graph bisection problem, and presented an intuitive graph for this structure as shown in Fig 3. We can see that this "globally convex" can also be considered as a global slowly varying structure superimposed with many local irregularities. The same structure has been found in circuit/graph partitioning and job-shop scheduling, etc..[11] Leary[12] also confirmed that there exist similar "funnel" structures in molecular conformation problems where the potential energy from the forces between atoms is minimized.

The issues described above are common in many practical optimization problems[13, 14]. For such class of problems, genetic algorithm[15] and simulated annealing[5] are the most common algorithms since they require little *a priori* information from the concerned problem and are generally applicable. However, these algorithms are designed for full-optimization and often lacking in efficiency. Controlled random search[3], i.e., Price algorithm, is also recommended for this situation[13, 16] but still suffers from lack of efficiency. In practice, these optimization algorithms are

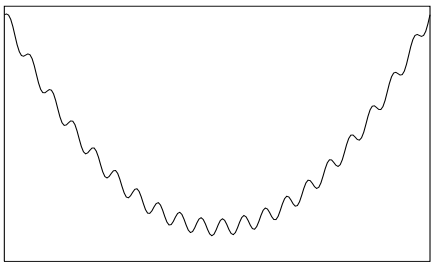

Figure 3: Big valley structure

often combined with local search techniques, such as, deepest descent and pattern search, to improve their efficiency. However, since these local search techniques use fixed local structures to guide the search process, they are usually susceptive to the effect of noises[17]. For example, in pattern search, the wrong pattern may easily be derived if the samples for pattern exploration are corrupted by noises. Furthermore, for the objective function with "globally convex" structures, local methods also perform inefficiently since there exist a large number of low-quality local optima. Multi-start local search algorithms will waste many efforts on examining these low-quality local optima and essentially work like an inefficient random sampling.

In contrast to most of the search algorithms, the RRS algorithm is mainly built upon random sampling. RRS performs the search process based on stochastic information on a certain sample area, therefore, its performance is less affected by noises. In addition, RRS is more efficient when dealing with the objective function with negligible parameters. This is because that random samples will still maintain its uniform distribution in the subspace composed of only those important parameters, and hence effectively removes negligible parameters from the optimization process. In this way, the efficiency of the search can be improved significantly. For the objective function with "globally convex" feature, the RRS is able to detect the overall structure approach global optima very quickly because of its overview-to-detail search strategy.

3 Design Ideas of Recursive Random Search

Random sampling is the simplest and most widely used search technique, which takes random samples from a uniform distribution over the parameter space. Despite its simplicity, random sampling is able to provides a strong probabilistic convergence guarantee, i.e., the optimization result converges to the global optimization with probability 1. Furthermore, random sampling has surprisingly proved to be more efficient than deterministic exploration methods, such as, grid covering, in terms of some probabilistic criteria and it is especially so for high-dimensional problems[18]. The disadvantage of random sampling is its apparent lack of efficiency. However, we will show that it is in fact *very efficient in its initial steps and its inefficiency is from the latter sampling*. In the following, we first describe the initial high-efficiency property of random sampling, the basis of the Recursive Random Search algorithm, and then present the basic idea of RRS.

3.1 Efficiency of Random Sampling

Given an measurable objective function $f(\mathbf{x})$ on the parameter space D with a range of $[y_{min}, y_{max}]$, we can define the *distribution function* of objective function values as:

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

where $y \in [y_{min}, y_{max}]$ and $m(\cdot)$ denotes *Lebesgue measure*, a measure of the size of a set. For example, *Lebesgue measure* is area in a 2-dimensional space, and volume in a 3-dimensional space, and so on. Basically, the above equation represents the portion of the points in the parameter space whose function values are smaller than a certain level y. $\phi_D(y)$ is a monotonously increasing function of y in $[y_{min}, y_{max}]$, its maximum value is 1 when $y = y_{max}$ and its minimum value is $m(x_*)/m(D)$ where x_* is the set of global optima. Without loss of generality, we assume that $f(\mathbf{x})$ is a continuous function and $m(\mathbf{x} \in D | f(\mathbf{x}) = y) = 0, \forall y \in [y_{min}, y_{max}]$, then $\phi(y)$ will be a monotonously increasing continuous function with a range of $[0, 1]$. Assuming a $y_r \in [y_{min}, y_{max}]$ such that $\phi_D(y_r) = r, \quad r \in [0, 1]$, a *r-percentile* set in the parameter space D can be defined:

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

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

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

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. Figure 4 shows the *r-percentile* set that n steps of random sampling can reach with a probability of 99%. We can see that *random sampling is highly efficient at initial steps since r decreases exponentially with increasing n, and its inefficiency is from later samples*. As shown in Figure 4, it takes only 44 samples to reach a point in $A_D(0.1)$ area, whereas all future samples can only improve r value of $\mathbf{x}^n_{(1)}$ at most by 0.1.

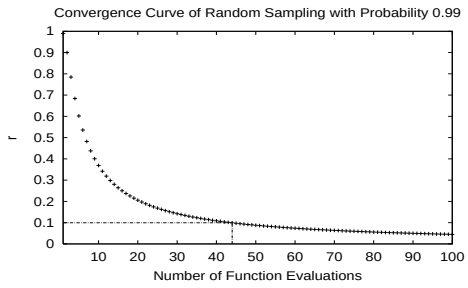

Figure 4: $A_D(r)$ of $x^n_{(1)}$ in random sampling with probability 0.99

3.2 Overview of The RRS Algorithm

The basic idea of RRS is to maintain the initial efficiency of random sampling by "restarting" it before its efficiency becomes low. However, unlike the other methods, such as hillclimbing, random sampling cannot be restarted by simply selecting a new starting point. Instead we accomplish the "restart" of random sampling by *changing its sample space*. Basically, we perform random sampling for a number of times, then move and resize the sample space according to the previous samples and start another random sampling in the new sample space. Given a black-box objective function, a desired optimization process should start with inspecting macroscopic features of the objective function, and then look further into microscopic features in selected promising areas. The search process of RRS algorithm

is fully consistent with this idea. In the beginning of the search, RRS performs sampling from the whole parameter space and thus examines the overall structure of the objective function. With the search continuing and the sample space gradually shrinking, the search gets more and more details of the objective function until it finally converges to a local optimum.

A stochastic search algorithm usually comprises two elements: *exploration* and *exploitation*. Exploration examines the macroscopic features of the objective function and aims to identify promising areas in the parameter space, while exploitation focuses on the microscopic features and attempts to exploit local information to improve the solution quickly. Various search techniques can be used for these two purposes. Since macroscopic features are hard to characterized, some unbiased search techniques, such as random search and random walk, are often used for exploration. Some algorithms also try to build a simple model to characterize the macroscopic features of the objective function and perform exploration based on this model. However, to choose an appropriate model for a certain problem is very difficult and require requires extensive *a priori* knowledge from the problem. Local search methods are the most commonly used techniques for exploitation, and hence exploitation is also called *local phase* in many literature and accordingly exploration is also known as *global phase*. Derivative-based local search methods, such as quasi-Newton method[19] and deepest descent[20], are very efficient for differentiable objective functions, however, they are not suitable for many practical problem because of its sensitivity to noises and limitation for the differentiability of objective function[13]. Direct search, such as Nelder-Mead simplex method[21] and pattern search[22], do not exploit the derivative of the objective function and are more suitable for the concerned problems.

Basically, the RRS algorithm uses random sampling for exploration and recursive random sampling for exploitation. Ideally it should only execute the exploitation procedure in promising areas. However, it is difficult to determine which areas are more promising and should be exploited. Many algorithms, such as multistart type algorithms, do not differentiate areas and hence may waste much time in trivial areas. Our approach is to identify a certain *r-percentile* set $A_D(r)$ and only start exploitation from this set. In this way, most of trivial areas will be excluded from exploitation and thus the overall efficiency of the search process can be improved. This can be illustrated by the example shown in Figure 5. The left graph shows a contour plot of a 2-dimensional multi-modal objective function and the right graph

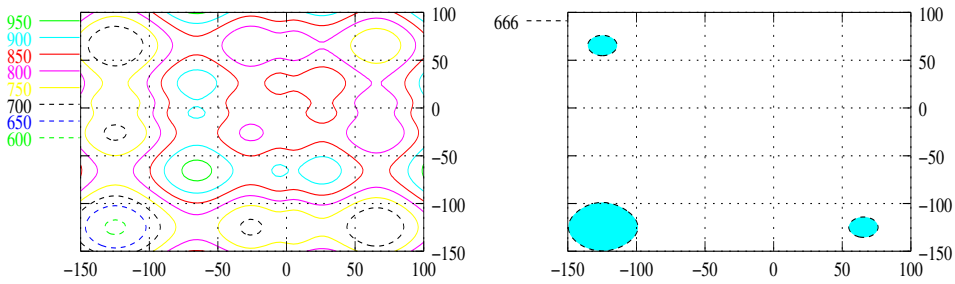

Figure 5: Contour plot of an objective function(left) and its region of $A_D(0.05)$(right)

shows the set of $A_D(0.05)$. As shown in the figure, the function has many local optima; however, only three regions of attraction remain in $A_D(0.05)$ (shaded areas in the right plot). Each of these regions 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 [7]. If we perform random sampling on the whole parameter space, the samples falling in $A_D(r)$ are also uniformly distributed over $A_D(r)$, consequently, they are more likely to belong to the region containing the global optimum. That means, if exploitation is started from these points, the search will arrive at the global optimum with a larger probability than other non-global optima.

It is desirable that the size of $A_D(r)$ region identified by exploration is as small as possible such that most of trivial areas are filtered out. On the other hand, its smallest size is limited by the efficiency of random sampling, i.e.,

it should be within the reach of initial high-efficiency steps of random sampling so that identifying a point in it will not take too long to lower the overall efficiency.

4 Algorithm Details

The basic idea of the RRS algorithm is to use random sampling to explore the whole parameter space and only start exploitation, i.e., recursive random sampling, for those points which fall in a certain $A_D(r)$ 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.

4.1 Exploration

In the exploration phase, random sampling is used to identify a point in $A_D(r)$ for exploitation. The value of r should be first chosen. Based on this value and a predefined confidence probability p, the number of samples required to make $Pr(\mathbf{x}^n_{(1)} \in A_D(r)) = p$ can be calculated as(according to Equation 4): $n = \frac{\ln(1-p)}{\ln(1-r)}$ (line 1 in pseudo-code). The algorithm uses the value of $f(\mathbf{x}^n_{(1)})$ in the first n samples as the threshold value y_r(line 4) and any future sample with a smaller function value than y_r is considered to belong to $A_D(r)$. In later exploration, a new $\mathbf{x}^n_{(1)}$ is obtained every n samples and y_r is updated with the average of these $\mathbf{x}^n_{(1)}$ (lines [21-23]). Note that this calculation of y_r is not intended to be an accurate estimation of the threshold for $A_D(r)$, instead it only function as the adjustment for the balance between exploration and exploitation. In other words, it is to ensure that on the average the exploration process will not continue for n samples and hence enter its low-efficiency phase.

In this exploration method, the confidence probability p should choose a value close to 1, for example, 0.99. The value of r decides the balance between exploration and exploitation and should be chosen carefully as discussed before. According to the current experience, we have used $r = 0.1$ and $p = 0.99$ in the algorithm, and with such values it only takes 44 samples to find a point for the estimation of y_r.

4.2 Exploitation

As soon as exploration finds a promising point $\mathbf{x_0}$ whose function value is smaller than y_r, we start a recursive random sampling procedure in the neighborhood $N(\mathbf{x}_0)$ of $\mathbf{x_0}$. The initial size of $N(\mathbf{x}_0)$ is taken as the size of $A(r)$, i.e., $r \cdot m(D)$, where D is the original parameter space since $\mathbf{x}_0$ belongs to $A(r)$ with a high probability. Currently a simple method is used to construct $N(\mathbf{x}_0)$: assume the parameter space D is defined by the upper and lower limits for its ith element, $[l_i, u_i]$, the neighborhood of $\mathbf{x}_0$ with a size of $r \cdot m(D)$ is the original parameter space scaled down by r, i.e., $N_{S,r}(\mathbf{x_0}) = \{z \in S \mid |z_i - x_{0,i}| < r^{1/n} \cdot (u_i - l_i)\}$(line 10), where $x_{0,i}$ is ith element of $\mathbf{x}_0$ and z_i ith element of $\mathbf{z}$. With this new sample space $N_{S,r}(\mathbf{x_0})$, random sampling is continued. And then based on the obtained samples, the sample space is re-aligned or shrunk as exemplified in Figure 6 until its size falls below a predefined level s_l, which decides the resolution of the optimization.

Re-align sub-phase As described above, exploitation first starts 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. Let's define an expected value of $\phi_{N(\mathbf{x}_0)}(f(\mathbf{x}_0))$, υ, 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 within 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 *re-align* operation. For example, in Figure 6, the exploration identifies a promising point C_1 and then the exploitation (i.e., random sampling) start in the neighborhood R_1 of C_1. After a few samples, a new point C_2 is found to be better than C_1, hence the sample space is moved from R_1 to the neighborhood R_2 of C_2. In this

Algorithm 1: Recursive Random Search

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

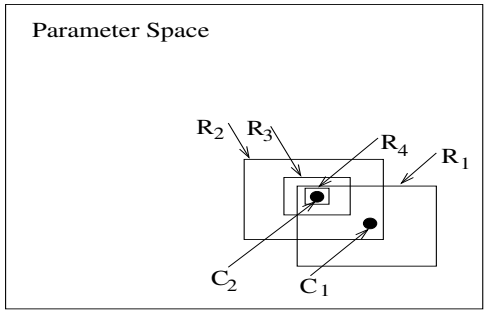

Figure 6: Shrink and Re-align Process

way, even if the initial $N(\mathbf{x}_0)$ (i.e., R_1 in the example) might miss the local optimum, the later re-align moves will still lead the search to converge to the local optimum.

Shrink sub-phase If random sampling fails to find a better point in l samples, that suggests $\phi_{N(\mathbf{x}_0)}(f(\mathbf{x}_0))$ is smaller than the expected level v. 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]). This is called *shrink* operation, which is performed only when we *fail* to find a better point in l samples. When the size of sample space is reduced to a value such that $\phi_{N(\mathbf{x}_0)}(f(\mathbf{x}_0))$ is larger than v, then "re-align" will take over again to moves to the local optimum. With re-align and shrink alternately performed, the sample space will gradually converge to the local optimum. For example, in Figure 6, after l unsuccessful samples in R_2, the sample space is shrunk to R_3, then to R_4 if sampling in R_3 continue to fail. The exploitation process continues until the size of sample space falls below a certain threshold, whose value is dependent on the resolution requirement of the optimization problem.

5 Tests on Standard Benchmark Functions

As discussed before, the design objectives of RRS are: high efficiency, scalability to high-dimensional problems, robustness to function evaluation noises and capability of handling negligible parameters. This section will present the performance tests of RRS in these aspects. A suite of classical benchmark functions have been used in our performance tests, such as, Square Sum, Rastrigin[23] and Griewangk[7], most of which have a large number of local optima and are considered very difficult to optimize. For example, Fig 7 shows the 2-dimensional version of one benchmark function, Rastrigin function.

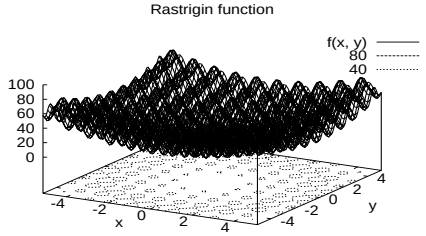

Figure 7: 2-dimensional Rastrigin function

The performance of a search algorithm is usually measured by examining the number of function evaluations to obtain a point satisfying a certain goodness criterion. Since the emphasis of our design objective is not on full optimization but achieving high efficiency in the limited time frame, we have used the convergence curve of the

optimization, i.e., the optimization result as a function of the number of function evaluations, to compare the efficiency of algorithm especially in the initial part of the optimization process. Basically, we execute the search algorithm for a certain number of function evaluations and draw the convergence curve. The performance of the algorithms are compared based on these convergence curves.

5.1 Tests on Efficiency of RRS

In the efficiency tests, the performance of RRS is compared with two popular search algorithms: controlled random search and multistart pattern search. Controlled random search is recommended for black-box optimization problems in many literature[24, 13]. Multistart type algorithms have also been one of the most popular methods in practice[14] and been demonstrated to work very well and outperform many more sophisticated algorithms, such as genetic algorithm and simulated annealing, in many practical problems[25, 26, 27]. Pattern search[22] is one of local search techniques which are usually recommended for black-box optimization[28].

In the tests, the search algorithms are executed on each function with dimension varying from 20 to 2000. To eliminate randomness caused by stochastic elements in the search algorithms, each test is repeated for 50 times with random starting points and the average of the results is used. The following parameters for the RRS algorithm have been used in the tests: $p = 0.99, r = 0.1, c = 0.5, \upsilon = 0.8, q = 0.99, s_t = 0.001$. Fig 8 shows one set of test results for Rastrigin function. Complete results on all benchmark functions are presented in [29]. The results show that the RRS algorithm performs much more efficiently than the other two search algorithms. Controlled random search is more like pure random search in the beginning of the search. From the results, we can see that it does perform very efficiently at its initial few steps and is better than multistart pattern search. However, with the search continuing, its performance quickly degrades since random sampling loses its efficiency in later sampling. The results also demonstrated that multistart pattern search cannot perform well for the objective function with a large number of local optima.

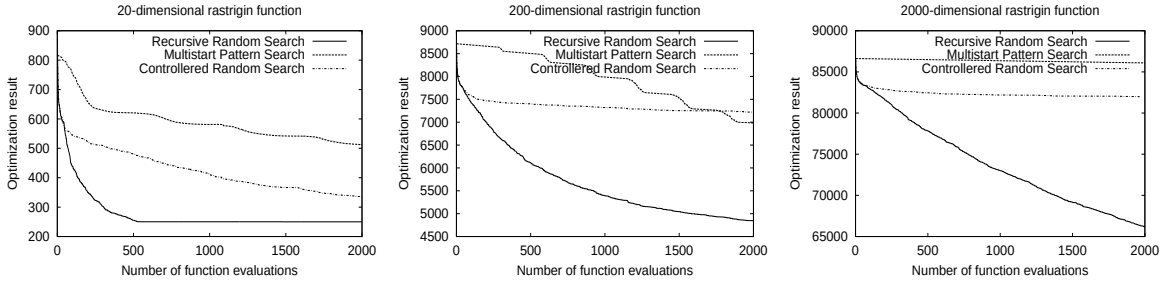

Figure 8: Performance tests on Rastrigin function

5.2 Tests on Noise-Resistance of RRS

Compared with local search methods, random sampling is more robust to the effect of noises. This section will compare the performance of RRS and multistart pattern search algorithm for noise-affected objective functions. Directly imposing random noises on the objective function may introduce the randomness into test results. Therefore, to obtain consistent results, Rastrigin function have been used to emulate the situations where the evaluation of the objective function is affected by small noises. Rastrigin function is defined as:

$$f(\mathbf{x}) = n \cdot A + \sum_{i=1}^{n} (x_i^2 - A \cdot \cos(2\pi x_i)) \tag{6}$$

It can be also considered as a simple sphere function $\sum_{i=1}^{n} x_i^2$ superimposed with the noise term $\sum_{i=1}^{n} A \cdot \cos(2\pi x_i)$. The magnitude of noises is determined by the value of A. To test the noise-resistance of the search algorithms, we vary

the noise level in Rastrigin function, i.e., the value of A, and see how the search algorithms perform under different magnitudes of noises. Note that the noise magnitude should not be too large to distort the overall structure of the original function. Figure 9 shows the test results on Rastrigin functions with different noise level and different dimensions. The results demonstrate that increasing magnitude of noises seriously degrade the performance of multistart pattern search while the effect on RRS is slight.

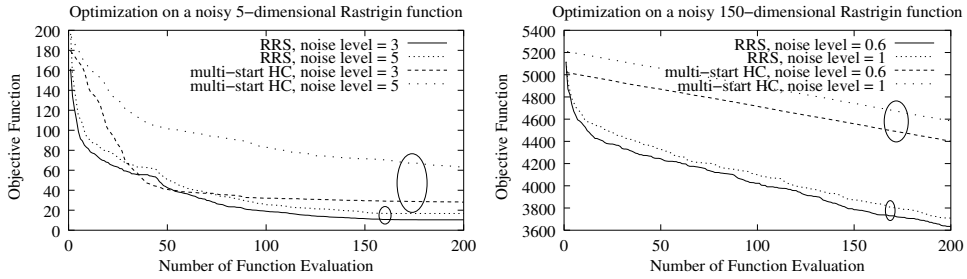

Figure 9: Noise-resistance tests of search algorithms

5.3 Tests on Objective Functions with Negligible Parameters

To simulate the situation of negligible parameters, an n-dimensional test function in Equation (7) is used:

$$f(\mathbf{x}) = \sum_{i=1}^{5} x_i^2 + 10^{-12} \cdot \sum_{i=5}^{n} x_i^2 \tag{7}$$

where $-500 < x_i < 500, i = 1 \ldots n$. In this function, the first five parameters essentially determine the function value while the others are trivial parameters with little effect. The tests are performed for the cases where there are 0, 5 and 10 negligible parameters, and the performances of RRS and multistart pattern search are compared. Fig 10 shows the test results. It can be seen that the introduction of trivial parameters has little effect on the performance of RRS while the performance of multistart patter search degrades considerably with increasing number of trivial parameters.

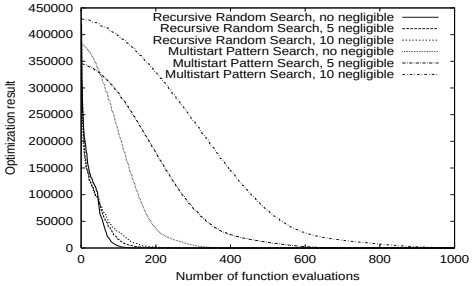

Figure 10: Performance tests on objective functions with negligible parameters

6 Application to Parameter Optimization of Network Protocols

The RRS algorithm has been successfully applied to the adaptive configuration of serveral important network protocols. For example, the parameters of Random Early Dectection(RED) algorithm have been tuned to maintain its stability in varying network conditions and achieve high utilization as well low queueing delay. And we have also

tuned `local_ref` attributes of a BGP domain to achieve the load balancing objectives for outbound traffic. These applications are presented in detail in [30]. This section summarizes one of them as an example, i.e., traffic engineering by tuning OSPF routing metrics.

Traffic engineering is a very important aspect in networking research. The objective of traffic engineering is to distribute the offered traffic load evenly across the network such that network resources are optimally utilized. In current Internet, IP traffic is mapped onto the network by standard routing protocols. However, these routing protocols normally do not take into account current network conditions and Quality of Service(QoS) constraints. As a result, the routing generated by these algorithms tend to generate a highly uneven mapping of traffic. That is, some links may get very congested and the other may be consistently underutilized.

Suppose the offered traffic load in a network is defined by a demand matrix, where the row index represents the source, the column index is the destination and the element of the matrix is the offered load from the source to the destination. The demand matrix can be obtained through network measurement or estimation. A routing algorithm will decides the network path taken by the traffic from a source to a destination. In this way, the traffic load represented by the demand matrix is mapped to the network. Open Shortest Path First(OSPF) is the *de facto* standard routing protocol for the intra-domain traffic, i.e., the traffic transfered within the same network domain. In OSPF algorithm, each network link is assigned with a link weight and the routing decision is completely based on the values of the link weights i.e., it routes the traffic from source to destination through the path with the minimum total link weight. Traditionally, the link weights in OSPF are set heuristically without considering QoS requirements. With the knowledge of the demand matrix, the OSPF link weights can be tuned to achieve traffic engineering objectives. This tuning problem has been demonstrated to be NP-hard [31] can be tackled with a black-box optimization approach.

To formulate the black-box optimization problem for OSPF link weight setting, an optimization objective, i.e., the performance metric of the network, has to be defined first. Consider a network composed of n links. For one link l_i, we calculate its link cost ϕ_i based on the following formula:

$$\phi_i(x_i) = \begin{cases} 1 & \text{for} \quad 0 \leq x_i < 1/3 \\ 3 & \text{for} \quad 1/3 \leq x_i < 2/3 \\ 10 & \text{for} \quad 2/3 \leq x_i < 9/10 \\ 70 & \text{for} \quad 9/10 \leq x_i < 1 \\ 500 & \text{for} \quad 1 \leq x_i < 11/10 \\ 5000 & \text{for} \quad 10 \leq x_i \end{cases} \tag{8}$$

where x_i is the normalized traffic load distributed on link l_i. Then, the objective of this optimization problem is to minimize the total link cost

$$\Phi = \sum_{i=1}^{n} \phi_i \tag{9}$$

This performance metric has been used for the design of AT&T WorldNet backbone network[31]. It heavily penalizes the routing causing congestion. Consequently the optimization of this metric will result in a good routing which evenly distributes traffic and voids link congestion.

Let $\mathbf{w}$ denote the link weight vector, whose ith element is the link weight for link l_i. For each specific $\mathbf{w}$, network simulation is run to evaluate the objective function Φ. Given a parameter space for $\mathbf{w}$, optimization can be performed to obtain the optimal or near-optimal solution of $\mathbf{w}$. Fig 11 shows the convergence curve of RRS on the ARPANET network which has 140 link weights. For comparison, we also show the convergence curve of a tabu-enhanced multistart hillclimbing search algorithm used in [31]. The straight line in the figure indicates the performance metric when one of heuristic configuration methods, the unit link weight, is used. The results show that the RRS algorithm performs very efficiently in the test. If we use the performance of the heuristic setting as the benchmark level, we can see that RRS finds a solution better than the heuristic setting with around 62.4% fewer function evaluations than the compared algorithm. Besides the results presented here, simulations on some other networks, such as MCI network, also show similar results.

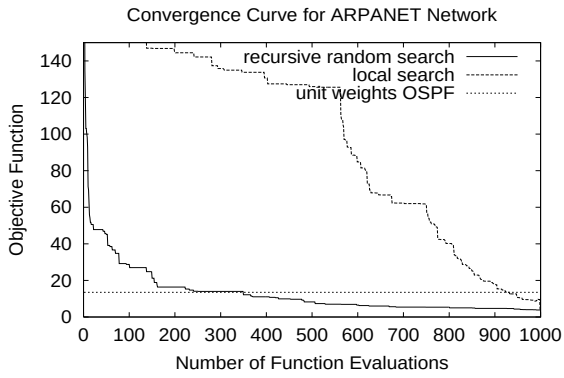

Figure 11: Convergence curves of optimization algorithms for ARPANET OSPF link weight setting

Besides the performance metric presented in this paper, we also attempted other metrics, such as, overall packet loss rate, whose optimization is especially important for TCP traffic load. In practice, the operator can formulate arbitrarily complex objectives according to their needs and request RRS to work on it to give the best-answer-so-far within a time bound.

7 Conclusion

This paper presented a new heuristic search algorithm, Recursive Random Search, which is designed for on-line network configuration optimization problems where the efficiency and noise-robustness of the algorithm are highly emphasized. In contrast to most other search algorithms, the new algorithm is mainly based on random sampling and does not include any traditional local search methods which are not scalable to high-dimensional problems and sensitive to the effect of noises. By constantly restarting random sampling with adjusted sample spaces, high efficiency at initial steps of random sampling can be maintained.

The test results on a suite of benchmark functions have shown that the RRS algorithm performs very efficiently, and is more robust to the effect of noises than the other algorithms in comparison. It is also demonstrated that RRS is of great advantage when dealing with the problems with negligible parameters. The RRS algorithm has been successfully used to the on-line tuning of several network protocols, such as RED, BGP and OSPF. However, its potential applications are unlimited and the employment on other network protocols is under investigation.

References

[1] Ratul Mahajan, David Wetherall, and Tom Anderson. Understanding bgp misconfiguration. In *Proceedings of ACM SIGCOMM*, 2002.

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

[3] W. L. Price. Global optimization by controlled random search. *Journal of Optimization Theory and Applications*, 40:333–348, 1978.

[4] J. H. Holland. *Adaptation in Natural and Artificial Systems*. Univ. of Michigan Press: Ann Arbor, 1975.

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

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

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

[8] T. C. Hu, V. Klee, and D. Larman. Optimization of globally convex functions. *SIAM Journal on Control and Optimization*, 27(5):1026–1047, 1989.

[9] K. D. Boese, A. B. Kahng, and S. Muddu. On the big valley and adaptive multi-start for discrete global optimizations. Technical Report TR-930015, UCLA CS Department, 1993.

[10] K. D. Boese, A. B. Kahng, and S. Muddu. a new adaptive multi-start technique for combinatorial global optimizations. *Operation Research Letters*, 16:101–113, 1994.

[11] Justin A. Boyan and Andrew W. Moore. Learning evaluation functions to improve optimization by local search. *Journal of Machine Learning Research*, 1(2000):77–112, 2000.

[12] Robert H. Leary. Global optimization on funneling landscapes. *Journal of Global Optimization*, 18(4):367–383, December 2000.

[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] Zelda B. Zabinsky. Stochastic methods for practical global optimization. *Journal of Global Optimization*, 13:433–444, 1998.

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

[16] V. P. Plagianakos and M. N. Vrahatis. A derivative-free minimization method for noisy functions. In P.M. Pardalos A. Migdalas and R. Burkard, editors, *Advances in Combinatorial and Global Optimization*, pages 283–296. 2001.

[17] 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.

[18] A. H. Kan and G. T. Timmer. Stochastic global optimization methods part I: Clustering methods. *Mathematical Programming*, 39:27–56, 1987.

[19] W. C. Davidon. Variable metric method for minimization. *SIAM Journal on Optimization*, 1:1–17, 1991. The article was originally published as Argonne National Laboratory Research and Development Report May 1959(revised November 1959).

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

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

[22] R. Hooke and T. Jeeves. Direct search solution of numerical and statistical problems. *Journal of the ACM*, 8(2):212–229, April 1961.

[23] 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.

[24] M. Ali, C. Storey, and A. Törn. Application of stochastic global optimization algorithms to practical problems. *Journal of Optimization Theory and Applications*, 95(3):545–563, 1997.

[25] 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.

[26] 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.

[27] 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.

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

[29] A recursive random search for optimizing network protocol parameters. Technical report, 2002.

[30] Automatic network performance management with on-line simulation. Technical report, 2002.

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