A Recursive Random Search Algorithm for Network Parameter Optimization

Tao Ye [1]   Shivkumar Kalyanaraman [2]

Abstract

This paper proposes a new heuristic search algorithm, Recursive Random Search(RRS), for black-box optimization problems. Specifically, this algorithm is designed for the dynamical parameter optimization of network protocols which emphasizes on obtaining good solutions within a limited time frame rather than full 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 basis on random sampling, the RRS algorithm is robust to the effect of random noises in the objective function and it performs especially efficiently when handling the objective functions with negligible parameters. These properties have been demonstrated with the tests on a suite of benchmark functions. The RRS algorithm has been successfully applied to the optimal configuration of several network protocols. One application to a network routing algorithm is presented.

1 Introduction

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

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

where D is the predefined parameter space, usually a compact set in $\mathbb{R}^n$. In these problem, 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, 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.

In the area of network engineering, the configuration of network protocol can also be formulated as a black-box optimization problem. The configuration of network protocols is widely considered a black art and is normally performed based on network administrators' experience, trial and error, etc.. These manual methods are often error-prone and not scalable to large complex networks. The on-line simulation system has been proposed in[24] to tackle this problem with a black-box optimization approach. As shown in Figure 1, the on-line simulation system continuously monitors network conditions and provides network models for simulation. With network simulation to evaluate the performance of a certain configuration, the optimization can be performed to search for a good configuration under current network conditions. For such network parameter optimization problems, the following features are usually present.

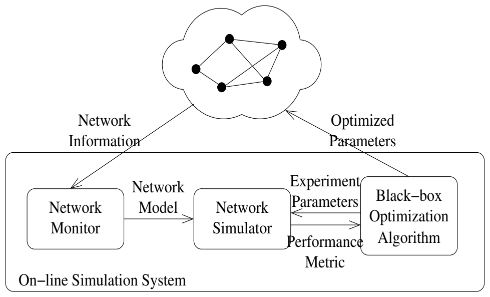

Figure 1: On-line simulation framework for network protocol optimization

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 might be very time-consuming. This also requires a highly efficient search algorithm to 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 all OSPF link weights of this network are to be configured, there will be thousands of parameters present in the optimization. High-dimensional optimization problems are usually much more difficult to

[1]Rensselaer Polytechnic Institute, Troy, New York 12180
[2]Rensselaer Polytechnic Institute, Troy, New York 12180

solve than low-dimensional problems because of "curse of dimensionality"[21].

Noise is often introduced into the evaluation of objective function since network simulation may be 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. Figure 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.

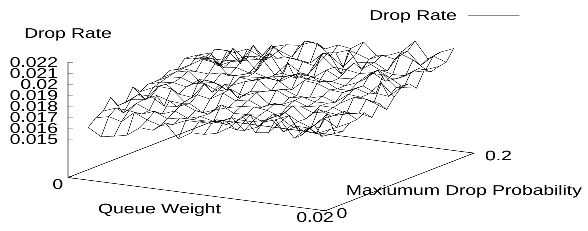

Figure 2: An empirical objective function obtained with network simulation (RED queueing management)

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 be significantly improved.

These issues are also very common in many other practical optimization problems[6, 26]. For such class of problems, genetic algorithm[17], simulated annealing[15] and controlled random search[18] are the most common optimization algorithms since they require little *a priori* information and are generally applicable. However, these algorithms are often lacking in efficiency. In practice, these stochastic algorithm are often combined with local search techniques to improve their efficiency. However, since these local search techniques exploit local structures to decide the search direction, they are usually susceptive to the effect of noises[19]. For example, in pattern search, the wrong pattern may easily be derived if the samples for pattern exploration are corrupted by noises.

In [25], we proposed a new search algorithm, Recursive Random Search(RRS), to tackle network parameter optimization problems efficiently. This paper is a extended version of [25]. The major feature of RRS is its basis on random sampling. Random sampling is usually considered a robust but inefficient technique. However, as we describe in this paper, random sampling is actually very efficient in its initial steps. The RRS algorithm exploits this feature by truncating the current random sampling and starting another one with the adjusted sampling space. Since RRS does not employ any local search method, it is also more robust to noises. Furthermore since random samples will still maintain its uniform distribution in the subspace composed of only those important parameters, RRS is able to effectively removes negligible parameters from the optimization process.

The above features of the RRS algorithm have been validated by tests on a suite of benchmark functions. The test results show that RRS outperforms the other algorithms in comparison. We have successfully applied the RRS algorithm to the configuration of several network protocols, such as buffer management algorithm and routing algorithm. One of them is presented in this paper.

The rest of this paper is organized as follows: in Section 2, we discuss design concepts of the RRS algorithm. In Section 3, we describe the details of this algorithm. In Section 4, the algorithm is tested on a suite of benchmark functions, and its performance compared with other stochastic algorithms. In Section 5, we apply the algorithm to one real network optimization problem, the configuration of OSPF routing algorithm. Finally, we conclude this paper in Section 6 and discuss further research directions.

2 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. Although it is a rather simple technique, 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[14]. The disadvantage of random sampling is its apparent lack of efficiency. However, we will show that random sampling 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, which is the basis of the Recursive Random Search algorithm and then present the basic idea of RRS.

2.1 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:

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

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* in a 2-dimensional space is just area, 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 $m(\mathbf{x}^*)/m(D)$ where $\mathbf{x}^*$ 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 3 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 3, 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.

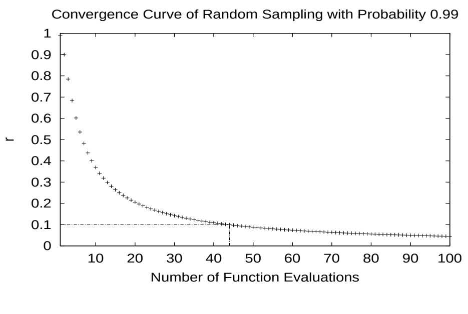

Figure 3: $A_D(r)$ of $x^n_{(1)}$ in random sampling with probability 0.99

2.2 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.

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 attempt 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, unbiased search techniques, such as random search and random walk, are often used for exploration. Some algorithms also tries to build a simple model to characterize the macroscopic features of the objective function and perform exploration based on this model. However, it is often difficult to choose an appropriate model for a practical problem and it 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*. As discussed in [6], although derivative-based local search methods, such as quasi-Newton method[7] and deepest descent[3], are very efficient for differentiable objective functions, they are not suitable for the problems considered in this paper because of its sensitivity to noises and requirement for the differentiability of objective function. Direct search methods, such as Nelder-Mead simplex method[16] and pattern search[11], do not exploit the derivative of the objective function and are more suitable for the concerned problems. But still, they are susceptible to the effect of noise since they perform the search based on a local structure.

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 4. The left graph shows a contour plot of a 2-dimensional multimodal objective function and the right graph 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

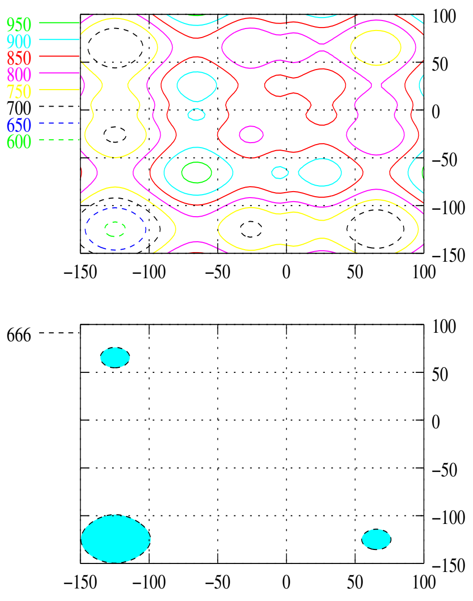

Figure 4: Contour plot of an objective function(left) and its region of $A_D(0.05)$(right)

for many optimization problems since the region of attraction containing the global optimum usually has the largest size [21]. 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.

3 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.

3.1 Exploration

In the exploration phase, random sampling is used to identify a point in $A_D(r)$ for exploitation. The value of r should

Algorithm 1: Recursive Random Search

```
1  Initialize exploration parameters $p$, $r$, $n \leftarrow \ln(1-p)/\ln(1-r)$ ;
2  Initialize exploitation parameters $q$, $\upsilon$, $c$, $s_t$, $l \leftarrow \ln(1-q)/\ln(1-\upsilon)$;
3  Take $n$ random samples $x_i$, $i = 1 \dots n$ from parameter space $D$;
4  $\mathbf{x}_0 \leftarrow \arg\min_{1\le i\le n}(f(\mathbf{x}_i))$, $y_r \leftarrow f(\mathbf{x}_0)$, add $f(\mathbf{x}_0)$ to the threshold set $\mathbf{F}$;
5  $i \leftarrow 0$, $exploit\_flag \leftarrow 1$, $\mathbf{x}_{opt} \leftarrow \mathbf{x}_0$;
6  while stopping criterion is not satisfied do
7      if $exploit\_flag = 1$ then
           // Exploit flag is set, start exploitation process
8          $j \leftarrow 0$, $f_c \leftarrow f(\mathbf{x}_0)$, $\mathbf{x}_l \leftarrow \mathbf{x}_0$, $\rho \leftarrow r$;
9          while $\rho > s_t$ do
10             Take a random sample $\mathbf{x}'$ from $N_{D,\rho}(x_l)$;
11             if $f(\mathbf{x}') < f_c$ then
                   // Find a better point, re-align the center of sample space to the new point
12                 $\mathbf{x}_l \leftarrow \mathbf{x}'$, $f_c \leftarrow f(\mathbf{x}')$;
13                 $j \leftarrow 0$;
               else
14                 $j \leftarrow j+1$;
               endif
15             if $j = l$ then
                   // Fail to find a better point, shrink the sample space
16                 $\rho \leftarrow c \cdot \rho$, $j \leftarrow 0$;
               endif
           endw
17         $exploit\_flag \leftarrow 0$, update $\mathbf{x}_{opt}$ if $f(\mathbf{x}_l) < f(\mathbf{x}_{opt})$;
       endif
18     Take a random sample $\mathbf{x}_0$ from $S$;
19     if $f(\mathbf{x}_0) < y_r$ then
           // Find a promising point, set the flag to exploit
20         $exploit\_flag \leftarrow 1$;
       endif
21     if $i = n$ then
           // Update the exploitation threshold every $n$ samples in the parameter space
22         Add $\min_{1\le i\le n}(f(\mathbf{x}_i))$ to the threshold set $\mathbf{F}$;
23         $y_r \leftarrow \text{mean}(\mathbf{F})$, $i \leftarrow 0$;
       endif
24     $i \leftarrow i+1$;
   endw
```

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.

3.2 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 5 until its size falls below a predefined level s_l, which decides the resolution of the optimization.

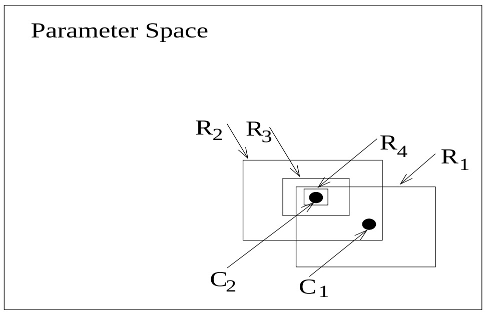

Figure 5: Shrink and Re-align Process

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 5, 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 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 υ. 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 υ, 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 5, 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 whole 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.

3.3 Remarks on the RRS Algorithm

The ideal behavior of a search algorithm is to first inspect the macroscopic features of the objective function, and then examine microscopic features with selected areas. The search process of RRS algorithm is completely consistent with this idea. In the beginning of the search, the sample space starts with the whole parameter space and the overall structure is examined. With the search continuing, the sample space gradually shrinks and consequently the search obtains increasingly detailed microscopic information until it finally converges to a local optimum. The search algorithms deviating the above ideal behavior usually cannot perform very well. For example, multistart type algorithms start a local search from each of random samples without considering the macroscopic features, therefore, their performance often suffers greatly from the time wasted on trivial areas.

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.

4 Tests on Standard Benchmark Functions

As discussed before, the design objectives of Recursive Random Search 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 the RRS algorithm in these aspects. A suite of classical benchmark functions have been used in our performance tests, such as,Rastrigin's function[20], Rosenbrock's Saddle[23], Griewangk's function[21] Ackley's function[1]. Most of these functions have a large number of local optima and are considered very difficult to optimize.

Usually, the benchmark tests of a search algorithm are performed by examining the number of function evaluations required to obtain a point close to the global optimum. Since the emphasis of our design objective is not on full optimization but achieving high efficiency in the limited time frame, we have adopted another method to compare the efficiency of algorithms. Basically, we execute the search algorithm for a certain number of function evaluations, and draw the convergence curve of the optimization, i.e., the optimization results as a function of the number of function evaluations. The performance of the algorithms are compared based on these convergence curves. To eliminate randomness caused by stochastic elements in the search algorithms, each test in the following is repeated for 50 times with random starting points and the average of the results is used.

4.1 Tests on Efficiency of RRS

First the RRS algorithm is tested on the benchmark functions in different dimensions and its performance is compared with two other search algorithms: controlled random search and multistart pattern search. Controlled random search is recommended for black-box optimization problems in many literature[2, 6]. Multistart pattern search is chosen because multistart type algorithms are always one of the most popular methods in practice[26] and have been demonstrated to work very well and outperform many more sophisticated algorithms, such as genetic algorithm and simulated annealing, in many practical problems[4, 12, 13]. Pattern search[11] is one of direct search techniques which are usually recommended for black-box optimization problems[22].

In the tests, the search algorithms are executed on each function with the function dimension varying from 20 to 2000. We have used the following parameters for the RRS algorithm: $p = 0.99, r = 0.1, c = 0.5, \upsilon = 0.8, q = 0.99, s_t = 0.001$. The test results for each benchmark function are shown in Fig 6-10. It can be seen that the RRS algorithm converges very rapidly and its efficiency is much better than the other two search algorithms. Note that in Figure 9, for 200-dimensional Ackely function, the performances of RRS and the multistart algorithm are close. However, RRS tends to converge to the optimal solution quickly while the convergence curve of multistart algorithm tends to be flatten out. In fact, in the subsequent search process which is not shown in the graph, it has been observed that RRS will perform much better than the other two algorithms. Controlled random search performs much like pure random search in the beginning when it has not yet converged to high-quality solutions. 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 and falls far behind the other two algorithms.

We also test the RRS algorithm on some low-dimensional classical benchmark functions[21] for which the optimization results approach the global optima with only hundreds of function evaluations. Similar convergence curve results to above can be obtained. We summarize the test results in Table 1 which shows the best-so-far function values after 75 function evaluations. It can be observed that RRS always delivers better results than the other two algorithms.

Function	Best-so-far Function Values		
	CSR	Multistart PatternSearch	RRS
Shekel5	-0.68	-0.34	**-1.97**
Shekel7	-0.77	-0.39	**-1.77**
Shekel10	-1.03	-0.45	**-1.92**
Hartman3	-3.57	-3.17	**-3.75**
Hartman6	-2.02	-1.77	**-2.60**
GoldPrice	25.75	587.87	**12.39**
CamelBack	-0.774	-0.114	**-0.994**

Table 1: Benchmark test results showing better optimization results found by RRS

4.2 Tests on Noise-Resistance of RRS

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 randomness into the 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

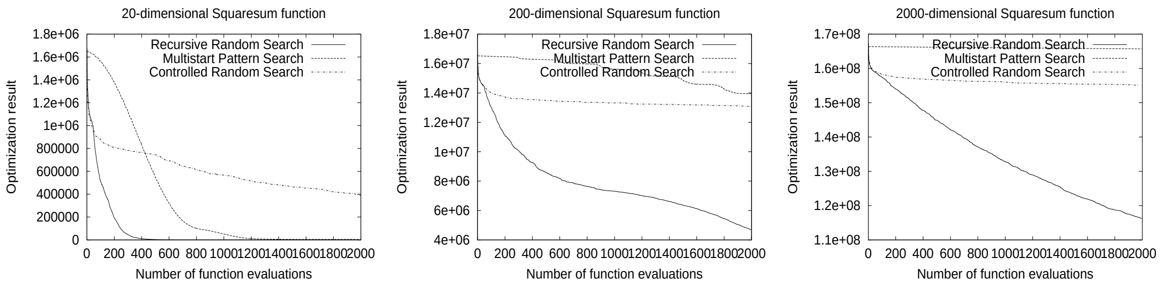

Figure 6: Performance tests on SquareSum function

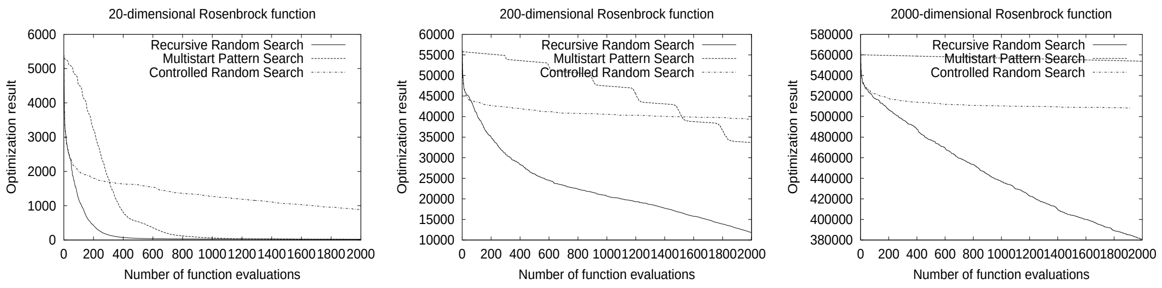

Figure 7: Performance tests on Rosenbrock function

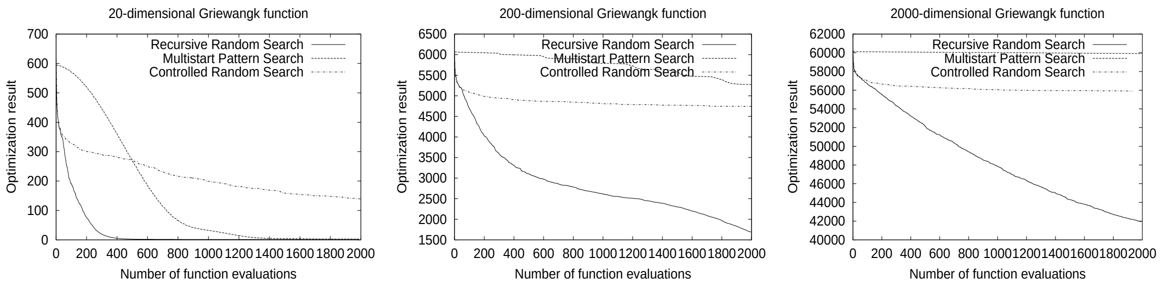

Figure 8: Performance tests on Griewangk function

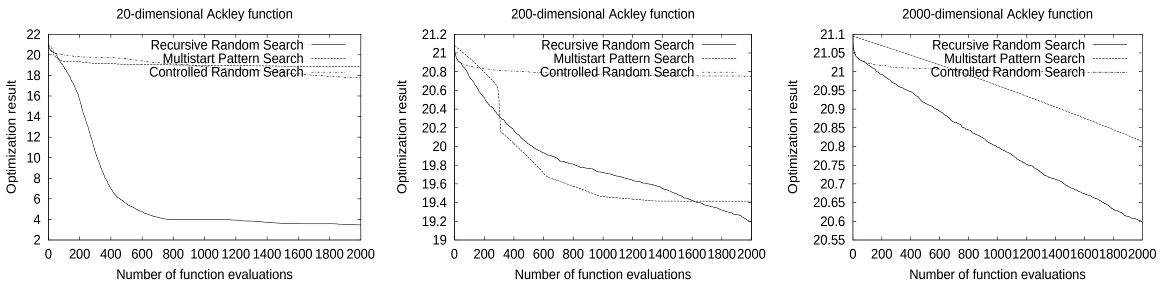

Figure 9: Performance tests on Ackley function

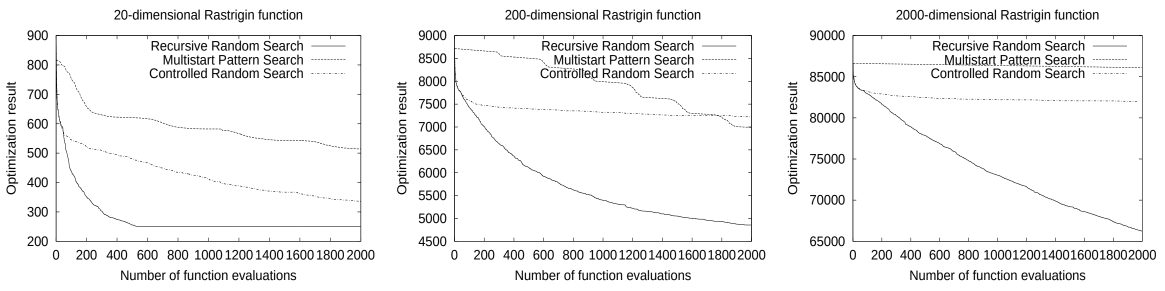

Figure 10: Performance tests on Rastrigin function

function is defined as:

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

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 11 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.

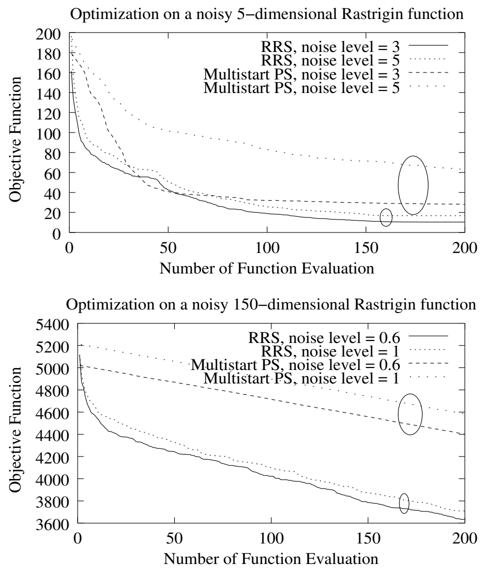

Figure 11: Noise-resistance tests of search algorithms

4.3 Tests on Objective Functions with Negligible Parameters

To simulate the occasion with trivial parameters, an n-dimensional test function in Equation (7) is used:

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

where $-500 < x_i < 500, i = 1 \ldots n$. In this function, the first five parameters are the major ones that determine the function value while the others are trivial parameters. The tests are performed for the cases where there are 0, 5 and 10 negligible parameters in the function, and the performances of RRS and multistart pattern search are compared. Figure 12 shows the test results. It can be seen that the introduction of trivial parameters can hardly affect the performance of the RRS algorithm while the performance of multistart pattern search degrades considerably with increasing number of trivial parameters. Therefore, the tests demonstrate that the RRS algorithm is able to automatically exclude negligible parameters from the optimization process and thus greatly improve the efficiency.

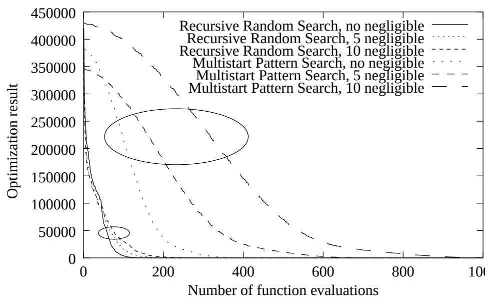

Figure 12: Performance tests on objective functions with negligible parameters

5 Application to Parameter Optimization of Network Protocols

The RRS algorithm has been successfully applied to the adaptive configuration of several network protocols to improve network performance. This section will present one example application, i.e., traffic engineering by tuning a routing algorithm configuration.

Traffic engineering is a very important aspect in networking research. Its objective 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. When routing the traffic, the routing algorithms used in these routing protocols 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. Some links may get very congested since most of traffic go through it and the other may be underutilized. This has been observed in many traffic measurements[5, 8].

Suppose the offered traffic load in a network is defined by a demand matrix, where the row index represents the source, the column index the destination and the element of the matrix the offered load from the source to the destination. Such demand matrix can be obtained through network measurement. 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 intra-domain traffic, i.e., the traffic transfered within the same network domain. OSPF is a *topology-driven* routing protocol and does not consider Quality of Service constraints in routing decision. In OSPF, each network link is assigned with a link weight and the traffic is routed from source to destination along 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 problem has been demonstrated to be NP-hard [9] and can be tackled with a black-box optimization approach. To formulate the black-box optimization for OSPF link weight setting, an optimization objective, i.e., the performance metric of the network, has be defined first. Since packet drop rate is a good indication to the congestion in the network and also has significant impacts on the performance of some major internet protocols, such as TCP, we have used the minimization of the total packet rate for a network as the optimization objective. Consider a network composed of n links, for link $l_i, i = 1 \dots n$, we calculate its packet drop rate ϕ_i based on the traffic load distributed to this link with the concerned OSPF link weight setting. Then the objective of this optimization problem is to minimize the total packet drop rate:

$$\Phi(\mathbf{w}) = \sum_{i=1}^{n} \phi_i(\mathbf{w}) \tag{8}$$

where $\mathbf{w}$ 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 its performance metric $\Phi(\mathbf{w})$. Given a parameter space for $\mathbf{w}$, optimization can be performed to obtain a good configuration of $\mathbf{w}$. Figure 13 shows the convergence curve of RRS on a large-scale network, EXODUS, which has 1040 links (parameters). For comparison, we also show the convergence curve of a tabu-enhanced multistart hillclimbing search algorithm used in [9]. The straight line in the figure indicates the performance metric when one of heuristic configuration methods, the unit link weight, is used. As shown in the graph, the RRS algorithm performs very efficiently. If we use the performance of the heuristic setting as the benchmark level, we can see that RRS finds a better solution than the heuristic setting with around 50% fewer function evaluations than the compared algorithm.

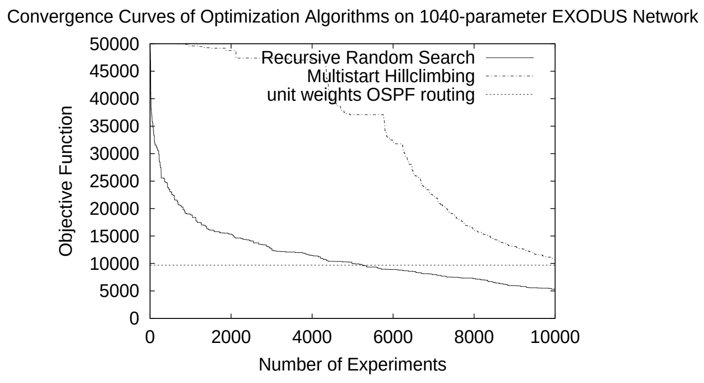

Figure 13: Convergence curves of optimization algorithms for EXODUS network OSPF link weight setting

Besides the OSPF link weight setting, the RRS algorithm has also been used for other network protocols, such as, Random Early Detection(RED) buffer management, Border Gateway Protocol(BGP). The details of these applications are presented in [10].

6 Conclusion

This paper has presented a new heuristic search algorithm, Recursive Random Search, which is designed to perform efficiently for simulation-based black-box optimization problems. Especially, the new algorithm is targeted for network configuration optimization problems where the efficiency is greatly emphasized. In other words, these problems would like a good result within a limited time. In addition, the evaluation of the objective functions of these problems is often affected by inaccuracies in simulation. The RRS algorithm is designed with consideration of these situations. In contrast to most other search algorithms, the new algorithm is mainly based on random sampling and does not includes any traditional local search methods which are usually not scalable to high-dimensional problems and sensitive to the effect of noises. The algorithm is tested on a suite of benchmark functions and compared with multistart pattern search algorithm and controlled random search algorithm. The results have shown that the RRS algorithm performs very efficiently, and is more robust to the effect of noises. It is also demonstrated that RRS is able to exclude negligible parameters from the optimization process and hence significantly improve the efficiency. The RRS algorithm has been successfully used in several network protocol configuration problems, such as queuing management algorithm RED, routing algorithms BGP and OSPF. One of the applications, OSPF tuning, is presented in this paper as an example.

References

[1] D. H. Ackley. *A Connectionist Machine for Genetic Hillclimbing*. Boston: Kluwer Academic Publishers, 1987.

[2] 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.

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

[4] 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.

[5] S. Bhattacharyya, C. Diot, J. Jetcheva, and N. Taft. Pop-level and access-link-level traffic dynamics in a tier-1 pop. In *ACM SIGCOMM Internet Measurement Workshop*, November 2001.

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

[7] 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).

[8] W. Fang and L. Peterson. Inter-as traffic patterns and their implications. In *Proceedings of Global Internet 99*, Rio, Brazil, 1999.

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

[10] T. Y. H. T. Kaur and S. Kalyanaraman. Minimizing packet loss by optimizing ospf weights using online simulation. In *Proceedings of IEEE MASCOTS 2003*, Orlando, FL, Oct. 2003.

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

[12] 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.

[13] 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.

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

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

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

[17] M. Mitchell. *An Introduction to Genetic Algorithms*. The MIT Press, 1996.

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

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

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

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

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

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

[24] T. Ye and et al. Traffic management and network control using collaborative on-line simulation. In *Proc. of IEEE ICC'01*, Helsinki, Finland, 2001.

[25] T. Ye and S. Kalyanaraman. A recursive random search algorithm for large-scale network parameter configuration. In *Proceedings of ACM SIGMETRICS 2003*, San Diego, CA, June 2003.

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