LARGE-SCALE NETWORK PARAMETER CONFIGURATION USING ON-LINE SIMULATION FRAMEWORK

By

Tao Ye

A Thesis Submitted to the Graduate

Faculty of Rensselaer Polytechnic Institute

in Partial Fulfillment of the

Requirements for the Degree of

DOCTOR OF PHILOSOPHY

Major Subject: Electrical Engineering

Approved by the
Examining Committee:

_____
Shivkumar Kalyanaraman, Thesis Adviser

_____
Biplab Sikdar, Member

_____
Christopher D. Carothers, Member

_____
Aparna Gupta, Member

Rensselaer Polytechnic Institute
Troy, New York

March 2003

LARGE-SCALE NETWORK PARAMETER CONFIGURATION USING ON-LINE SIMULATION FRAMEWORK

By

Tao Ye

An Abstract of a Thesis Submitted to the Graduate

Faculty of Rensselaer Polytechnic Institute

in Partial Fulfillment of the

Requirements for the Degree of

DOCTOR OF PHILOSOPHY

Major Subject: Electrical Engineering

The original of the complete thesis is on file
in the Rensselaer Polytechnic Institute Library

Examining Committee:

Shivkumar Kalyanaraman, Thesis Adviser

Biplab Sikdar, Member

Christopher D. Carothers, Member

Aparna Gupta, Member

Rensselaer Polytechnic Institute
Troy, New York

March 2003

CONTENTS

ABSTRACT vi

1. Introduction 1
   1.1 Network Performance Management 1
   1.2 Black-box Optimization 2
   1.3 Overview 4
2. Network Management 8
   2.1 Network Management Model 8
      2.1.1 Organization Model 9
      2.1.2 SNMP-based Network Management 10
      2.1.3 Functional Model 12
   2.2 Automatic Network Performance Management 13
      2.2.1 Optimization-based Performance Management 13
      2.2.2 On-line Simulation Scheme 14
3. Search Algorithm Review 16
   3.1 Structure of Stochastic Search Algorithms 17
      3.1.1 Exploration Methods 18
         3.1.1.1 Random Sampling 18
         3.1.1.2 Random Walk 19
         3.1.1.3 Model Fitting 19
      3.1.2 Exploitation Methods 20
         3.1.2.1 Downhill Simplex Methods 21
         3.1.2.2 Hillclimbing 22
         3.1.2.3 Pattern Search 22
   3.2 Stochastic Search Algorithms 24
      3.2.1 Multi-start Algorithms 24
      3.2.2 Controlled Random Search 26
      3.2.3 Genetic Algorithm 27
      3.2.4 Simulated Annealing 28
      3.2.5 Tabu Search 29
   3.3 No Free Lunch Theorem 30

4. Design Issues of Efficient Search Algorithm . . . 33
4.1 Design Requirements for Network Optimization . . . 33
4.2 Distribution Function of Objective Function . . . 34
4.3 Efficiency of Search Algorithms . . . 35
4.3.1 Efficiency Measures of Optimization Algorithms . . . 36
4.3.2 Goodness Metric of Optimization Results . . . 39
4.3.3 Comparison of Some Algorithms with M_ϕ . . . 41
4.4 Large-scale Optimization Problems . . . 43
5. Recursive Random Search Algorithm . . . 48
5.1 Design Ideas of Recursive Random Search . . . 48
5.1.1 Efficiency of Random Sampling . . . 48
5.1.2 Balance between Exploration and Exploitation . . . 50
5.2 Algorithm Details . . . 51
5.2.1 Exploration . . . 51
5.2.2 Exploitation . . . 53
5.2.3 Remarks on the RRS Algorithm . . . 55
5.2.4 Efficiency of the RRS Algorithm . . . 56
5.3 Test Results . . . 58
5.3.1 Benchmark Functions . . . 59
5.3.2 Tests on Efficiency of RRS . . . 63
5.3.3 Tests on Noise-Resistance of RRS . . . 66
5.3.4 Tests on Objective Functions with Negligible Parameters . . . 66
6. Unified Search Framework . . . 68
6.1 The Unified Search Framework . . . 69
6.1.1 Sampler . . . 70
6.1.2 Memory . . . 73
6.1.3 Computing Resource Model and Management Mechanism . . . 74
6.1.4 Parallel Optimization Strategy . . . 77
6.2 Test Results . . . 78
6.2.1 Effect of Memory Types . . . 78
6.2.2 Effect of Resource Allocation Strategy . . . 79
6.2.3 Scalability of Parallel Optimization . . . 80
6.3 Conclusion . . . 82

7. Application to Network Optimization 83
7.1 Optimize RED for Network Congestion Control 83
7.1.1 Formulation of RED Optimization Problem 84
7.1.1.1 Parameter Sensitivity of RED 84
7.1.1.2 Optimization Objective 86
7.1.2 Simulation Results 88
7.1.2.1 Simulation for Optimization of Single RED 88
7.1.2.2 Real Network Experiment for Optimization of Multiple RED Queues 90
7.1.3 Conclusion 92
7.2 Optimize OSPF for Traffic Engineering 92
7.2.1 The Objective Function 94
7.2.1.1 Link Drop Probability 95
7.2.1.2 The Optimal General Routing 98
7.2.2 Optimization of OSPF Weights Using On-line Simulation . . . 99
7.2.3 Simulation Results 100
7.2.3.1 Comparison of Search Schemes 102
7.2.3.2 Heuristic Piecewise Linear Metric 103
7.2.3.3 Packet Drop Rate Metric 103
7.2.3.4 Optimizing OSPF for Improving Packet Drop Rate . 104
7.2.4 Conclusions 106
7.3 Optimize BGP Routing Algorithm 106
7.3.1 Traffic Demands 108
7.3.2 Optimal Routing Calculation for Load Balancing 109
7.3.3 The BGP Optimization Scheme 111
7.3.4 Simulation Results 112
7.3.4.1 Optimizing for Load Balancing 112
7.3.4.2 Minimizing Packet Loss 113
8. Conclusion and Future Research Direction 115
8.1 Conclusion 115
8.2 Future Research Direction 117
LITERATURE CITED 119

ABSTRACT

Today's Internet relies on a variety of important network protocols. The current parameter configuration process of these protocols is mainly manual and widely considered a black art. This thesis tackles this parameter setting problem by formulating it as a "black-box" optimization problem. In this approach, we use an on-line simulation system to monitor and simulate the network, and then use a black-box optimization algorithm to optimize the parameters of the concerned network protocol. This black-box approach allows flexibility in terms of objectives and metrics of the desired optimization and can be applied to a wide range of network protocols.

We first investigate the properties of the network protocol optimization problems and examine the applicability of various optimization techniques. For the concerned problems, the desired optimization algorithm is required to be highly efficient, scalable to high dimensions and robust to noisy objective functions. Based on these requirements, we propose a Recursive Random Search (RRS) algorithm whose major feature is its basis on random sampling. We empirically validate the advantages of RRS with extensive tests on a suite of benchmark functions and application in some real network optimization problems. To provide a more generally applicable solution for practical optimization problems, we also propose a Unified Search Framework (USF), which includes a variety of search techniques as building blocks. This framework can be used as the platform to build tailored optimization strategies by combining a selection of building blocks according to the features of the underlying problem. Furthermore, USF includes the mechanisms to parallelize search techniques and allocate available computing resources among them such that the resources are optimally utilized.

Finally, we investigate the configuration problems of several network protocols, such as, Random Early Drop (RED), Open Shortest Path First (OSPF) and Border Gateway Protocol (BGP). We formulate these problems into black-box optimization problems, some of which have thousands of parameters. We then apply the optimization techniques developed in this thesis to them. Simulations and experiments

have demonstrated the effectiveness and efficiency of the on-line simulation system and the proposed optimization techniques.

CHAPTER 1
Introduction

1.1 Network Performance Management

Internet is a global network whose operation relies on a variety of network protocols. Today's network protocols like BGP and OSPF were designed for one primary service: "best effort reachability." But now network operators want to deploy Virtual Private Networks(VPN), manage traffic within ASes to meet Service Level Agreements(SLA), and between ASes (at peering points) to optimize complex peering agreements. The designers of such protocols included "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 serve more functions[1]. The objective of network performance management is to automate network protocol configuration by introducing monitors and intelligence into the network. Though some network management tools are emerging to aid operators, a lot more needs to be done.

The on-line simulation system proposed in[2] is one such contribution to this important space. This system can 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. As illustrated in Fig 1.1, the basic idea of this system is to formulate network protocol configuration as a black-box optimization problem. With the network protocol considered as a black-box, network simulation can be used to evaluate its performance for various parameter settings. Based on this, an optimization algorithm can then be employed to find "good" protocol configurations for the current network conditions. The "black-box" approach allows flexibility in terms of objectives and metrics of the desired optimization, and hence can be applied to a variety of configuration problems. This approach can be used to manage any network protocol

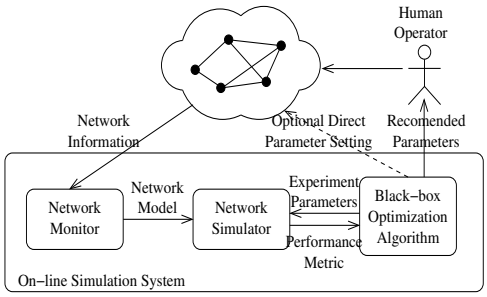


Figure 1.1: On-line simulation system for adaptive configuration of network protocols

as long as the protocol has *tunable* parameters whose setting has a substantial effect on network performance. An efficient optimization algorithm is essential to the on-line simulation system since network is a dynamic system and the optimization of network configuration should be quickly completed before significant changes in network conditions happen. The objective of this dissertation is to investigate the problem of network parameter optimization and design the efficient optimization strategies for this problem.

1.2 Black-box Optimization

In view of the complexity of network performance analysis, a variety of network simulation software, such as ns[3], $SSFnet$[4] and $GloMoSim$[5], have been developed to help understand behaviors of these protocols empirically. These simulation software enable us to evaluate the network performance for a certain parameter setting of a network protocol. As a result, an empirical mapping can be established between network performance and protocol parameter setting as shown in Figure 1.2. With this mapping, the optimal configuration of a network protocol can be formulate as a *black-box optimization problem*, where the network is considered as a black box and its performance is evaluated with network simulation. An optimization algorithm is then used to search the parameter space of the network protocol for the setting which delivers the optimal or near-optimal performance.

A general unconstrained optimization problem can be formulated as follows (assume minimization): given a real-valued objective function $f : \mathbb{R}^n \rightarrow \mathbb{R}$ which

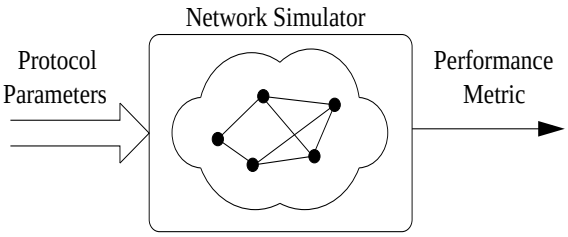


Figure 1.2: Empirical mapping between protocol parameters and network performance

maps a parameter setting to a performance metric, find a global minimum,

$$\mathbf{x}^* = \arg\min_{\mathbf{x}\in D} f(\mathbf{x}) \tag{1.1}$$

where D is called parameter space, usually a compact set in $\mathbb{R}^n$ which contains the global minimum as an interior point. In many practical optimization problems, the objective function $f(\mathbf{x})$ is analytically unknown and can be considered as a black-box, such as the one illustrated in Figure 1.2, and the information about this back-box can only be obtained by function evaluation with computer simulation or other indirect ways. Such class of problems are so-called *black-box optimization*. Since usually little *a priori* knowledge is known about the underlying "black-box", and these optimization problems are very hard to solve. Additionally the objective functions of these problems are often non-linear and multi-modal, which makes the problems even harder. This type of optimization is 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.

The optimization problem described above arises in many scientific and engineering areas. A large number of optimization algorithms have been proposed and successfully applied in practice, such as, multi-start hill-climbing[6], genetic algorithm[7] and simulated annealing[8]. In the vast amount of optimization application literature, few *analytical efficiency* results have been reported due to the complexity of optimization problems. *The comparison of optimization algorithms is mainly accomplished in an empirical way by experimenting on a somewhat arbitrary selection of benchmark objective functions*[9, 10]. So far, there has been no

consistent report on efficiencies of different algorithms. Furthermore, even choosing appropriate efficiency measures still remains a major research problem.

In fact, the No Free Lunch theorem[11, 12] has demonstrated that no matter what performance metric is used, the average performance of any optimization algorithm is the same. In other words, there is no generally efficient algorithm and no single algorithm can consistently perform better in every class of problems than the others. For one specific optimization problem, the most efficient algorithms are those which best exploit the available information of the objective function. Therefore, to design an efficient optimization algorithm, the properties of the underlying problem have to be first carefully examined. In addition, there is a tradeoff between efficiency and applicability. A desired optimization algorithm should strike an appropriate balance between these two aspects. The objective of this dissertation is to bridge the gap between black-box optimization and network configuration problems, find appropriate search techniques and strategies for the efficient optimization, and apply these techniques to solving real network configuration problems. Due to the intractable difficulty in the analytical examination of optimization algorithms, this dissertation also uses the qualitative empirical comparison approach as in most of optimization literature. The quantitative comparison is still an open question.

1.3 Overview

Many practical problems, such as, circuit design[13], job scheduling[14], molecular conformation[15, 16], have been solved successfully with optimization techniques. However, in the area of network performance management, optimization has not been used regularly. One important purpose of this dissertation is to bridge the gap between these two fields. We first examine the issues present in network optimization problems, then investigates the applicability of various search techniques. Among these issues, the most important ones are efficiency, scalability to high-dimensional problems and robustness to noisy objective function. Noting that random sampling is very efficient in initial steps and also very robust to noises, this dissertation designs the Recursive Random Search(RRS) algorithm which takes advantage of these features of random sampling to perform the efficient optimization

for network configuration problems.

RRS is designed to be a general optimization algorithm for network optimization problems which will deliver high efficiency in most cases. However, as stated in No Free Lunch theorem[12], there is no optimization algorithm which can consistently outperform the others in every class of problems. RRS may be outperformed in some cases by an optimization algorithm which aggressively exploiting the problem-specific properties. In fact, the design objective of RRS is to strike an appropriate balance between efficiency and applicability. To provide a general solution for practical optimization problems, this dissertation also proposes a Unified Search Framework(USF), which includes a variety of search techniques as building blocks and combine a selection of them based on the features of the underlying problem. Therefore, this framework can be used as the platform to build tailored optimization strategies for various optimization problems. Furthermore, USF includes a resource management mechanism which attempts to fully exploit the available computing resources and appropriately allocate them among parallel executed search methods. In practice, an optimization problem is often present with a certain amount of computing resources available for use by optimization. The objective of optimization can be considered as: given a certain amount of computing resources, achieve the optimization objective with the highest efficiency. USF includes a resource management and allocation mechanism to fulfill this objective.

Applying the on-line simulation scheme to real network performance management problems is another area investigated in this dissertation. When formulating the optimization problem for a network protocol, it is important to design an optimization metric to achieve the desired performance management objective. Many performance metrics exist, such as, network throughput, queueing delay and packet loss rate. Each of them measures one aspect of network performance. Sometimes, multiple metrics has to be considered in the optimization. For such occasion, appropriate multi-criteria optimization techniques should be used. In this dissertation, the on-line tuning problems of three important network protocols, i.e., RED queueing management, OSPF routing protocol and BGP routing protocol, are investigated and successfully solved with the on-line simulation scheme.

The rest of this dissertation is organized as follows:

Chapter 2 reviews general models of network management and important techniques in this area. Then we describe the on-line simulation scheme as the solution to automatic network performance management.

Chapter 3 reviews stochastic search algorithms for black-box optimization. An stochastic search algorithm normally comprise two elements: exploration and exploitation. We first reviews some important exploration and exploitation techniques which are appropriate for use in black-box optimization. Then we describe some popular stochastic search algorithms based on their exploration and exploitation techniques, and discuss their suitable problem class. Finally we describe the No Free Lunch Theorem.

Chapter 4 examines the issues in the design of an efficient algorithm for network optimization. To design an optimization algorithm, the properties of the underlying problems have to be first examined. In this chapter, we study the properties of network optimization problems. To compare the efficiency of different optimization algorithms, an appropriate performance measure should be used. This chapter also investigates different measures used in optimization literature and presents the correct methodology to compare algorithm efficiency.

Chapter 5 presents the design ideas and details of the Recursive Random Search algorithm. The efficiency of the RRS algorithm is validated empirically with benchmark tests and real applications. The test results on a set of standard benchmark function are presented. These tests are designed to examine the performance of RRS on efficiency, scalability, robustness to noises and handling negligible parameters.

Chapter 6 describes the Unified Search Framework (USF) which is designed to be a general solution for large-scale black-box optimization problems. We also discuss how to use USF as a flexible platform to integrate various techniques and allocate available computing resources among these techniques.

Chapter 7 presents the application of on-line simulation scheme to real network optimization problems. The optimization problems are formulated for three important network protocols, i.e., RED buffer management, OSPF routing protocol

and BGP routing protocol. Simulation results have demonstrated the effectiveness of the on-line simulation scheme on improving network performance.

Chapter 8 concludes the thesis and describes future research directions.

CHAPTER 2
Network Management

A network usually comprises a large number of network devices, network protocols and services. The normal operation of the network replies on the proper configuration of all these entities. Any failure or misconfiguration may result in performance degradation or service discontinuity of the network. Therefore, these network entities have to be carefully *managed* to maintain their working order. *Network management* means monitoring and controlling the network from a central location with the assistance of a variety of tools and devices. In the original phase of network management, network administrators have to remotely log on network devices to perform management tasks. As the sizes of networks grow rapidly in recent decades, there is an increasing need for the automatic network management architecture which is able to manage large-scale, heterogeneous networks. Various techniques have been proposed to achieve this objective. In this chapter, we will review the development in this area.

2.1 Network Management Model

In the 1980s, International Standardization Organization (ISO) completed the standardization of network management model with a serial of ISO standards[17]. According to the OSI/ISO standards, a network management architecture model comprises four models: organization model, information model, communication model and functional model.

Organization Model describes the components in a network management system and their relationships.

Information Mode defines the structure and storage of management information, and is usually divides into two parts, i.e., Structure of Management Information (SMI) and Management Information Base (MIB). MIB defines the variables residing in a managed device. These variables store the management

information and are used for information exchange between network management entities. SMI describes the syntax and semantics of the definitions in MIB and is defined by Abstract Syntax Notation One(ASN.1)[18], a formal language developed by CCITT and ISO to specify data types and structures for storage of information. The data definition with ASN.1 makes it independent of the lower-layer protocols in compunction model.

Communication Model handles how the management information is exchanged between network management entities.

Functional Model defines management functions to be implemented for various management objectives.

2.1.1 Organization Model

Figure 2.1 illustrates a general network management organization model. In

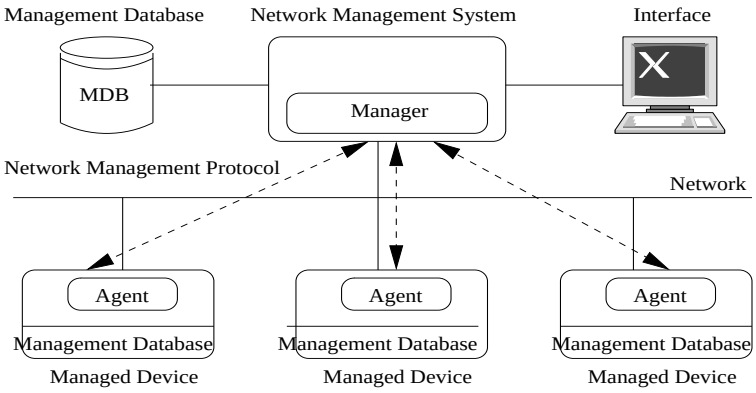


Figure 2.1: ISO network management organization model

this model, *network manager* is the control center of the network management system, which collects and analyze management information from various devices in the network to establishes an overall view of the network and perform the network management functions. Network manager also provides an interface for the network administrator to monitor and manually control the network. *managed devices* means the network devices controlled by network manager, which can be of any type, such as, routers, hubs and repeaters. On each of these devices, a software model called

management agent is run to collects and reports the management information of the managed device to network manager. Meanwhile, it also receives control signals from network manager and reacts on it correspondingly. The information exchange between management agents and network manager is performed through a certain *network management protocol*, such as, SNMP[19], CMIP[18].

The model shown in Figure 2.1 is essentially a two-tier centralized management system. For a very large network, centralized network management may introduce heavy extra traffic into networks and is not very reliable and efficient. Furthermore, the manager is the single failure point and may become the bottleneck of the whole system, hence the centralized scheme is lacking in scalability. With the rapid increase of network size and complexity, the decentralized management becomes necessary. In the *hierarchical* scheme, many local network managers used to manage local networks, and they are all managed by a super manager, i.e., *Manager of Manager* (MoM). In the *distributed* scheme, even the need for MoM is eliminated. Local network managers are no longer controlled by a single MoM, instead, they cooperate with each other to perform the management of the whole network.

2.1.2 SNMP-based Network Management

The network management protocol is the base of a network management system. A network management protocol usually covers both information model and communication model, i.e., it not only defines the communication protocol between management entities, but also defines SMI and MIB of the management system. Two most common network management protocols are: Simple Network Management Protocol(SNMP)[19] and Common Management Information Protocol(CMIP)[18]. CMIP is a comprehensive object-oriented standard proposed by ISO. It is highly complex and requires high consumption of system resources. A normal workstation may not afford enough resource to load a complete CMIP stack. These drawbacks make it not widely applicable and is mainly used in the complex and large-scale telecommunication systems, such as the Telecommunications Management Network (TMN)[20] of International Telecommunication Union (ITU).

Simple Network Management Protocol(SNMP)[19] is a much simpler protocol

compared with CMIP. The philosophy behind SNMP is to minimize the impact caused by adding the management function to a managed device. It is this simplicity philosophy that makes SNMP a great success in practice. In current Internet, SNMP has become the de facto standard for TCP/IP network management. SNMP is originally proposed in the 1970s as an interim solution to network management with the long-term objective of migrating to OSI standard CMIP, however, due to its enormous success, it seems that the long-term objective will never happen. Enhancement has been constantly added to make it independent of OSI standard. SNMPv2[21] adds some significant improvement to overcome the shortcoming of SNMP, such as, the efficient transferring of bulk data. It also allows the information exchange between managers, and hence make possible the distributed management architecture. Since no agreement can be reached in SNMPv2, one of the intended major enhancement, security consideration, is postponed to SNMPv3[22], which is basically a SNMPv2 with additional security and administration capabilities.

SNMP is essentially a request/response polling protocol with only five protocol messages. Normally an SNMP network manager issues queries to gather network management information from various SNMP agents. An SNMP agent can also send unsolicited information to the manager with a *trap* message when a predefined event occurs. The SNMP manager can configure the managed devices by setting their MIB variables. In addition to device-oriented management, Remote Network Monitoring (RMON)[23] is used to monitor the status of the network by inspecting network traffic of local network segments. In RMON, *probes* are introduced into local network segments which monitor and analyze network traffic statistics and send relevant information to network manager. Since probes monitor local network activities, it is more reliable and efficient than monitoring these devices remotely from the manager. Meanwhile, collected data is first processed locally and only relevant information is sent to the manager, as a result, the traffic incurred by network management can be reduced significantly. RMON is original designed to monitor link-layer data. And due its enormous success, RMON2[24] has extend its monitoring capability to higher layers, from the network layer to the application layer. This allows network manager to analyze traffic by protocols.

2.1.3 Functional Model

No matter what network management system is used, the essential purpose is to perform certain management functions. Functional model defines the functions provided by a network management system. Five functional areas are defined in ISO network management standard:

Performance Management monitors various performance measures of the network and controls the network so that the requirements for these performance measures are satisfied. Some common measures include, network throughput, link utilization, queueing delay, packets arrival jitter, etc.

Configuration Management collects information related to network configuration, such as network topology and software configuration, and control the initialization and modification of network configuration.

Accounting Management measures network utilization and use this information for billing purpose or access control to provide fair and optimal resources utilization among users.

Fault Management detects and logs the faults occurring on network devices, such as, device shut-down, link break-off. It also tries to fix these faults.

Security Management controls the access to various network resources and prevents the breach of security rules. Security management establishes the mapping between network resources and users, and restricts the access of users to appropriate resources according to this mapping. Security management comprises authentication and authorization.

Among these functional areas, performance management is very important for optimal utilization of network resources but may be the least implemented in current network management applications for its complexity. This dissertation will only consider the problems in network performance management.

2.2 Automatic Network Performance Management

In performance management, network performance metrics are monitored. Whenever a performance metric drops below a certain threshold, the alarm is sent to network manager and appropriate measures are taken to reconfigured the network and bring the network performance to normal. The essential task of performance management is to establish the correlation among network scenarios, configuration and performance metric, i.e., the relationship described by the following equation:

$$\mathcal{C} = f(\mathcal{N}, p) \tag{2.1}$$

where $\mathcal{N}$ denotes network scenario, p objective performance metric and $\mathcal{C}$ network configuration, for example, the parameters of a certain network protocol. Equation 2.1 basically derives the configuration $\mathcal{C}$ which is required to achieve the performance objective p in the scenario $\mathcal{N}$. Due to the complexity of the Internet, the analytical derivation of Equation 2.1 is usually not realistic. This is mainly performed through experiences of network administrators with specialized training. And performance management basically remains a monitoring and manual configuration system.

2.2.1 Optimization-based Performance Management

Automatic performance management requires the derivation of the correlation represented Equation 2.1. Some network management application packages have attempted to build an expert system which basically codes human experience into programs. However, this approach is not scalable because of the ever increasing network complexity and constantly addition of new network techniques. With the development of network modeling and simulation, it becomes possible to empirically examine network performance with network simulation software, such as *ns*[3], *SSFNET*[4]. In other words, we can obtain through network simulation the following empirical equation:

$$p = f^{-1}(\mathcal{N}, \mathcal{C}) \tag{2.2}$$

With Equation 2.2, given a certain objective performance metric p_0 and network scenario $\mathcal{N}_0$, it is possible to employ an optimization algorithm to search the parameter space of $\mathcal{C}$ for a certain $\mathcal{C}_0$ which satisfies $p_0 = f^{-1}_{empirical}(\mathcal{N}_0, \mathcal{C}_0)$. In this way, we can establish the automatic mapping between $\mathcal{C}$ and $\mathcal{N}, p$ as described in Equation 2.1. Essentially, this approach formulates the optimal network configuration as an "black-box" optimization problem for the lack of analytical $f^{-1}(\mathcal{N}, \mathcal{C})$ formula, and network simulation is used to empirically evaluate the network performance metric, i.e. the value of $f^{-1}(\mathcal{N}, \mathcal{C})$. Since network simulation can easily establish the empirical formula of $f^{-1}(\mathcal{N}, \mathcal{C})$ for any performance metric and any network scenario, the optimization-based performance management approach is very flexible. It can be used to optimize network configuration for any performance metric in any network scenario.

2.2.2 On-line Simulation Scheme

Despite the general applicability of the optimization-based approach, the previous efforts in this area mainly focus on the traditional areas of optimization applications, such as network topology design, routing and bandwidth allocation[25, 26, 27]. Only a few of them attempt to deal with the optimal configuration of network protocols[28]. Furthermore, these efforts only consider the optimization in the phase of network design, i.e, $\mathcal{N}$ is stationary, and is not aimed at performance management in varying network conditions. To address these problems, a general optimization-based performance management framework, the on-line simulation scheme, has been proposed[2], which can be easily adapted to the optimization of any network entity. As shown in Figure 2.2, the on-line simulator continuously monitor the managed network and control the configuration of a certain network protocol with SNMP. A typical on-line simulator comprises the following function units: *monitor and modeling*, *optimization algorithm* and *network simulation*, as shown in Figure 1.1 of the previous chapter. Whenever network conditions change, the optimization of the controlled network protocol is executed to maintain the network performance above the expected level. Thus, the on-line simulation scheme equips the network with pro-active, dynamic and automated management capabilities.

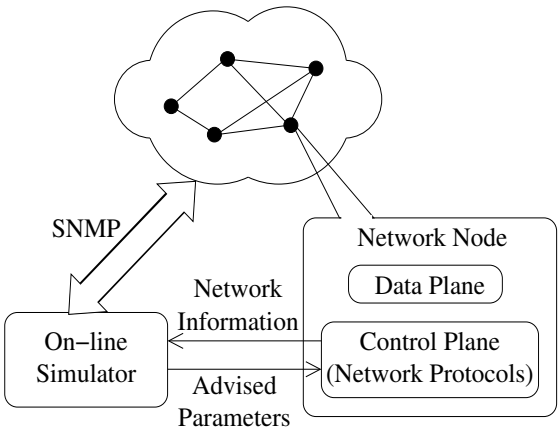


Figure 2.2: On-line simulation scheme

In the previous efforts which only consider the phase of network design, the optimization is executed off-line, and consequently the efficiency of the optimization algorithm is not a important factor as long as its effectiveness is guaranteed. In other words, as long as the optimization algorithm can finally find the optimal solution, the time consumption of the optimization is not important. In these cases, any general optimization algorithms, such as multi-start hill-climbing and genetic algorithms, can be used to perform the desired optimization. However, in the on-line simulation scheme, an efficient and reliable search scheme is essential to the success of on-line tuning since the underlying assumption is that network conditions are *quasi-stationary* and the optimization should be completed before significant changes happen. The objective of this dissertation is to examine the problems in designing an efficient optimization algorithm for such a system and provide the solutions to these problem.

CHAPTER 3
Search Algorithm Review

Given an objective function $f(\mathbf{x})$, its optimization is just a continuous sampling process from its parameter space D until the optimization objective is reached. A search algorithm d is just a mapping strategy which decides a new sample point $\mathbf{x}_{n+1}$ from previous samples and their function values $\{\mathbf{x}_i, f(\mathbf{x}_i)\}, i = 1 \ldots n$:

$$\mathbf{x}_{n+1} = d(\{\mathbf{x}_1, f(\mathbf{x}_1)\}, \{\mathbf{x}_2, f(\mathbf{x}_2)\}, \ldots, \{\mathbf{x}_n, f(\mathbf{x}_n)\}) \tag{3.1}$$

The search algorithms can be divided into two basic classes: *deterministic algorithms* and *stochastic algorithms*. Deterministic algorithms, such as, exhaustive enumeration and branch-and-bound[29], use deterministic mapping strategies in Equation 3.1, and usually provide an absolute guarantee of solution. However, these algorithms impose strict restrictions on the applicable problems, such as, requirements for Lipschitz constant, differentiability, and hence their application is very limited. On the other hand, stochastic algorithms, such as, controlled random search[30], genetic algorithm[31], simulated annealing[13], introduce stochastic methods into mapping strategies, as a result, can at most provide a probabilistic guarantee of convergence to the global optimum, i.e., when the search proceeds long enough, the probability of finding the global optimum tends to 1. Many of stochastic algorithms are pure heuristic and lacking in the support of theoretical analysis. In spite of these disadvantages, stochastic algorithms are very popular in practical optimization application since they impose much less restrictions and are much more widely applicable. Empirically, they have been demonstrated to be very successful in a wide range of scientific and engineering areas. This dissertation will only consider stochastic search algorithms.

3.1 Structure of Stochastic Search Algorithms

A stochastic search algorithm usually consists of two conflicting elements: *exploration* and *exploitation*. Exploration encourages the search process to examine unknown regions, while exploitation attempts to take advantage of structures of the objective function to find better solutions quickly. These two elements are combined in one search algorithm and the balance between them is maintained by a *balance strategy*. Figure 3.1 illustrates the relationship of these three elements in a search algorithm.

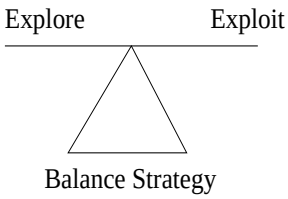


Figure 3.1: Exploration, exploitation and balance strategy

Exploration is also known as *global phase* in some literature[6, 32] and its objective is to provide a probabilistic guarantee of convergence to the global optimum for the search algorithm. That is, exploration mainly targets for the effectiveness of a search algorithm. For an exploration technique, the following theorem holds[6]: *Let the parameter space D be compact, then a sampling process converges to the global minimum of any continuous function $\Longleftrightarrow$ the sequence of sample points is everywhere dense in A.* In other words, the exploration should cover the whole parameter space when the search process continues for long enough time. An ideal exploration method would also identify the promising areas and visit these areas first. However, in practice, it is very difficult to achieve this objective.

Exploitation tries to exploit structural properties of the objective function to find better solutions quickly. Its objective is to improve the efficiency of the algorithm. Local search methods are the most widely used exploitation techniques, therefore, *exploitation* is often called *local phase* as well[6, 32]

Balance Strategy decides how the above two elements are correlated to each other in a search algorithm and coordinates their execution. The balance strategy can be either probabilistic or deterministic. For example, multi-start hill-climbing uses random sampling for exploration, a local search technique for exploitation and a deterministic balance strategy to execute the two procedures alternately.

Despite the vast number of search algorithms, the basic exploitation and exploration techniques used in these algorithms are not many. The rest of this chapter will first review the search techniques for exploration and exploitation, then describe some popular optimization algorithms in terms of their exploration, exploitation and balance strategies, and then discuss their applicability to optimization problems.

3.1.1 Exploration Methods

Basically, there are two kinds of exploration: biased and unbiased. Unbiased exploration doesn't differentiate regions in the parameter space while biased exploration tries to identify promising areas and sample them with higher probabilities. Although biased methods seems more desirable for efficiently exploring the parameter space, practically, it is hard to decide the correct bias, i.e., correctly identify promising areas.

3.1.1.1 Random Sampling

Random sampling is the simplest and most widely used exploration technique. All multi-start type of algorithms use this exploration method. Random sampling takes random samples according to a uniform distribution over the parameter space and hence is an unbiased method. The following convergence property can be proved[33]:
Let $y_n^{(1)}$ be the smallest function value found in n random samples, If $f(x)$ is continuous, then $y_n^{(1)}$ converges to the global minimum value y_* with probability 1 (almost surely) with increasing n, i.e.,

$$P[\lim_{n\to\infty} y_n^{(1)} = y_*] = 1. \tag{3.2}$$

In spite of its simplicity, 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[32].

3.1.1.2 Random Walk

Random walk is an exploration method originated from local search methods. Suppose a neighborhood can be defined for each point in the parameter space, random walk explore the parameter space by randomly choosing a point from the neighborhood of the current point, moving to the new point and repeating the process. Depending on the structure of neighborhood, this method may have a strong bias towards local areas. The convergence property of this method is also dependent on the definition of the neighborhood.

3.1.1.3 Model Fitting

As mentioned before, an ideal exploration method should identify the promising areas and and sample these areas with higher probability. In practice, it is hard to decide which areas are more promising than other. Model fitting[6] is a biased exploration method which use a predefined model to decide which areas are more promising based on previous samples. The basic procedure can be described as follows:

1. choose an appropriate model according to *a priori* knowledge on the objective function.
2. fit the model, i.e., adjust the parameters of the model, to previous samples.
3. find the optimum point in the model and the area around this point is the promising area to be further exploited.

Here the optimum point in the model can be found analytically if the model is simple enough, otherwise, an optimization process has to be performed for the model. The rationale behind this is that the model is much more simple or cheaper to evaluate than the original objective function and hence it is much easier to optimize.

Otherwise, the burden of auxiliary optimization will become as heavy as the direct optimization on the original objective function.

Model fitting is usually considered as the most efficient techniques in terms of the number of function evaluations required for optimization since all previous samples are used to decide the next sample point. However,in practice, it is usually very difficult and require enough *a priori* knowledge to choose an appropriate model which is essential to the success of this method. On one hand, the model should be accurate enough to approximate macroscopic features of the objective function. On the other hand, it should also not be too complicated to make the computation intractable. Deterministic models, such as polynomial regression[34], are relatively simple and good for computational purpose. Statistical models[35, 36], such as Weiner process, seem more appropriate for complicated and highly oscillating objective functions. However, the computation cost involved is often very high, and its application is limited to low-dimensional problems with expensive objective functions, i.e., the function evaluation is a costly computation task. .

3.1.2 Exploitation Methods

Exploitation methods try to improve the efficiency of the search algorithm by taking advantage of structural properties of the objective function. Although it have been realized that complex practical problems can only be solved by exploiting their structures, not much advance has been made in this area for the complexity of the problems. And *local search* methods remain the most prevalent exploitation method in practice, which exploits the *local correlativity* in objective functions, i.e., points in a local area tend to have similar function values. Basically, a local search method searches for a better point in the neighborhood of the current point and repeat the process at the new point until a local optimum is reached. Based on the information used to search for better points in a neighborhood, local search methods can be classified as methods using high order derivative information, such as gradient $f'(\mathbf{x})$ or Hessian $f''(\mathbf{x})$, and methods using only function evaluation (zero-order derivative information), i.e., $f(\mathbf{x})$, which is called *direct search* methods. Higher order local search methods, such as Newton's methods[37], steepest decent[37], are usually

more powerful and are highly efficient when applicable. However, they require the knowledge of $f'(\mathbf{x})$ or/and $f''(\mathbf{x})$, which is often not available in black-box optimization. Therefore, alternative methods, such as quasi-Newton method[38], have been proposed which estimate derivative information by finite-differencing. However, in many practical problems, there often exist random noises in the objective functions, as a result, the estimated derivative information would easily be corrupted and thus misleading to the search. In addition, in some cases, the objective functions may even be non-differentiable. For these problems, direct search methods are often recommended and more widely used in practical black-box optimization. These methods directly compare the rank of function values and hence are less affected by noises[39, 40, 41]. In the following, we will review some important direct search methods.

3.1.2.1 Downhill Simplex Methods

Simplex is a geometric figure composed of $n+1$ point in a n dimensional space. Downhill simplex method is a direct search method first proposed by Spendly, Hext and Himswork[42]. Their observation was that it should take no more than $n+1$ points to decide a downhill move since $n+1$ values of $f(\mathbf{x})$ would be needed to estimate $\delta f(\mathbf{x})$ via finite-difference. The simplex method first takes an initial non-degenerate simplex from the parameter space, then move this simplex towards the local optimum by continuously *reflecting* the worst point in the centroid of the remaining points as illustrated in Figure 3.2. A non-degenerate simplex is one for

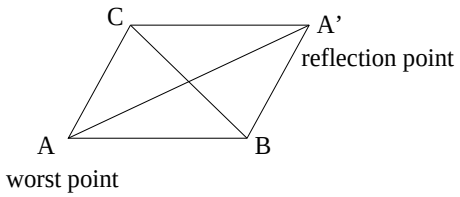


Figure 3.2: Reflection of simplex in 2-dimensional space

which the set of edges adjacent to any vertex in the simplex forms a basis for the space. This requirement is to make sure that any point in the parameter space can be constructed by taking linear combinations of the edges adjacent to any given vertex.

Nelder and Mead[43] improved the performance of the simplex method by adding two operations into the moves: *expansion* and *contraction*. The basic idea here is that when the search is moving down long inclined plane, the simplex is expanded to accelerate the move, while when the search comes close to the the neighborhood of a local optimum the simplex is contracted. Nelder-Mead simplex method has become the most popular method used for black-box local optimization problems[44, 45] and has been found very effective in many practical problems. However, Nelder-Mead simplex method is a pure heuristic method and its robust remains problematic. In fact, occasional convergence to a non-stationary point has been reported[46].

3.1.2.2 Hillclimbing

Hillclimbing is the most simple and straightforward local search technique. Suppose a neighborhood structure is defined in the parameter space, hillcimbing can be described as follows:

1. Take a sample x_0 from the parameter space D
2. Generate another sample x' from the neighborhood N of x_0, if $f(x') < f(x_0)$, replace x_0 with x'.
3. If a consecution of m samples have failed to generate a better point in the neighborhood of x_0, then terminate the process. Otherwise, repeat 2.

A good neighborhood structure and neighbor generation method are essential to the efficiency of this method.

3.1.2.3 Pattern Search

The original pattern search algorithm is first proposed by Hooke and Jeeves[47]. Torczon[48] presents a general pattern search model which also covers the class of similar algorithms such as coordinate search with fixed step sizes, evolutionary operation using factorial designs[49] and the multi-directional search algorithm[46]. According to Torczon's definition, pattern search methods proceed by conducting a series of exploratory moves about the current iterate and updating the associated information(pattern) which is used in future move to speed up the convergence to

the local optimum. These moves can be viewed as sampling the function about the current iterate in a well-defined deterministic fashion in search of a new iterate. The individual pattern search methods are distinguished, in part, by the manner in which these exploratory moves are conducted. By enforcing some limits on the pattern structure and search outcome, it has been proved that the pattern search is guaranteed to converge to a stationary point[48].

Figure 3.3 shows an example for the coordinate pattern search in a 2-dimensional space. In the figure, a black dot means a successful move, i.e., the new point is bet-

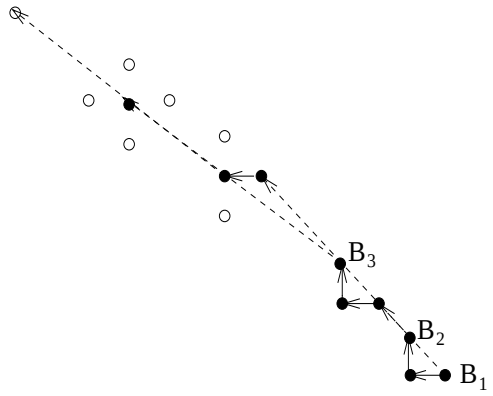


Figure 3.3: A pattern search example

ter than the old one while a empty dot means a unsuccessful attempt. Suppose the search starts with B_1, the pattern search will proceed as follows:

1. change x by one step size (increase or decrease), move to it if the new point is better.

2. change y by one step size, move to it if the new point is better.

3. project a new point according to the direction and the distance from B_1 to B_2, and move to it if the new point is better.

4. repeat 1, 2 and then project a new point according to the direction and the distance from B_2 to B_3.

5. Repeat the above procedure until no improvement can be achieved by exploratory moves and projected moves.

3.2 Stochastic Search Algorithms

With the few exploration and exploitation methods, a large number of search algorithms have been proposed by combining these methods with various balance strategies. This section will describe some most popular stochastic algorithms in practice and examine what exploration and exploitation methods have been used and how they are combined.

3.2.1 Multi-start Algorithms

Multi-start algorithms are based on *local search* methods. To avoid getting trapped in a local optimum, multi-start algorithms start a new search from another point whenever it reaches a local optimum. Usually random sampling is used to generate new starting points, i.e., exploration. However, model fitting techniques can also be used to generate starting points which more likely result in global optima. Such search algorithms are termed as adaptive multi-start algorithms[50, 34]. When the objective function does follow the conjectured model, model fitting can greatly improve the optimization efficiency. However, it is normally very difficult to find a proper model for a given problem.

Local search methods are used in multi-start algorithms for exploitation, such as, steepest ascent, pattern search, downhill simplex. The balance strategy of multi-start algorithm is a deterministic one which executes exploration and exploitation alternately. Since one run of exploitation will take much more time than one exploration, this balance strategy actually gives a very strong preference to the exploitation process.

Multi-start algorithms have been used widely for its simplicity and effectiveness. In practice these algorithms have produced excellent solutions in many practical problems, such as computer vision tasks[51]. It outperformed simulated annealing on the traveling salesman problem (TSP)[52], and outperformed genetic algorithms and genetic programming on several large-scale testbeds[53]. Multi-start algorithms are also attractive for their trivial parallelizability on the distributed computing architecture. One major disadvantage of multi-start algorithms is that it may waste a lot of time in examining unpromising areas when there are a large

number of local optima. This is due to its *strong preference to exploitation*. Normally a search algorithm should maintain a good balance between exploration and exploitation. One reason that genetic algorithm performs well in practice is its well-maintained balance[54] between exploration and exploitation. Furthermore, since local search methods base their search strategies on the information of a certain local structure, it may perform very badly when this information cannot be accurately estimated, for example, in the situation where the objective function is affected by noise. This is especially unfavorable for practical optimization problems where the objective function is evaluated with simulation. Since simulation often includes many approximations and inaccuracies, the accurate evaluation of the objective function is normally unavailable. In such situations, the objective function can be considered to be affected by noise and the performance of multi-start local search algorithms will suffer.

The term *region of attraction*(RoA) is used in optimization literature to denote a maximum set of points in the parameter space, any of which, when applied with the local search method, will lead to the same local optimum. A problem in multi-start algorithms is they may revisit the regions of attraction already examined! The efficiency will be greatly lowered if there are many revisit occurrences. Clustering methods[32, 55] are proposed to address this problem. The basic idea is to first take a number of random samples from the parameter space and then group these samples into a few clusters using a certain clustering analysis technique. Only one local search is executed for each cluster. It is hoped that each cluster belongs to only one region of attraction so that revisits can be avoided. However, it turns out to be very difficult to accurately form the clusters from random samples[55], especially for high-dimensional problems. One cluster may include multiple regions of attraction or samples from one region of attraction may be divided into multiple clusters. For the objective function with many local optima, the performance of clustering methods is similar to multi-start algorithms since the chances of revisits diminish.

3.2.2 Controlled Random Search

Controlled random search(CRS) is a population-based search algorithm. It is first proposed by Price[56] and also called Price's algorithm. CRS first randomly generates a population of samples from the parameter space, and then uses downhill simplex method to move the population towards the global optima. Suppose a n-dimensional objective function is to be optimized, the basic CRS algorithm can be described as follows:

1. Randomly generate a population of m points from the parameter space.
2. Randomly select $n+1$ points from the population and make a downhill simplex move.
3. If the new sample is better than the worst member in the population, then the worst member is replaced with this new sample.
4. Repeat the above process until a certain stopping criterion is satisfied.

In CRS, random sampling is used for exploration and downhill simplex for exploitation. Its balance strategy first executes exploration and then switches completely to exploitation. Since exploration is only performed in the beginning of the search, the convergence to the global optima is not guaranteed. The problem of getting trapped in a local extreme can be alleviated by using a large population or introducing new members into the population with random sampling during the search[40]. Despite of this disadvantage, CRS has proved to be very effective in practice and is widely used[40, 57, 9, 58]. In addition, since there is no local search method involved, CRS is more robust to noise in the objective function. Besides its failing to provide the convergence guarantee, another disadvantage is its low efficiency. Especially in the beginning, the population is composed of random samples and CRS essentially performs like a pure random sampling. Therefore, for many situations in practice where it is desired to obtain a good solution quickly, CRS may not be able to fulfill the objective.

3.2.3 Genetic Algorithm

Genetic algorithm(GA)[31, 7, 54] is another population-based search algorithm. It is inspired by the natural evolution process where the fitter members of a population survive. There are three important processes in GA: *selection*, *mutation* and *crossover*, which work together to accomplish exploration and exploitation. This algorithm maintains a certain number of sample points as a base population in each iteration. Then through *selection*, those high-quality members get survived to next step with higher probability. After *selection*, new members are introduced by alternating the survived members with unary operator *mutation* and binary operator *crossover*. *Mutation* randomly changes some of parameters for one member according to a mutation rate. *Crossover* combines parts from two different members to construct a new member.

The rationale behind GA is so-called "Building Block Hypothesis"[31, 7], i.e., the optimization problem can be broken down to many subproblems in lower dimension and the solution of the original problem can be obtained by combining the solutions (*building blocks*) of these subproblems. It should be noticed that this is rather different than most of other search algorithms which normally exploit the local correlativity of the objective function. Instead GA tries to exploit the parameter separability of the objective function. Basically, *mutation* is an exploration process which explores the parameter space for new building blocks and *crossover* is an exploitation process which combines available building blocks for better solutions. The balance strategy in GA doesn't have strong bias towards either exploration or exploitation. This may contribute to its success in many application[54].

Genetic algorithm has been successfully applied to many areas. It maintains a good balance between exploration and exploitation, and hence is able to achieve good performance in practice. The success of GA is also dependent on "Building Block Hypothesis", which can also be explained by *parameter separability*, i.e., the objective function can be decomposed into many functions in lower dimensions. Exploiting the parameter separability will substantially improve the optimization efficiency when handling high-dimensional problems with such a property. Therefore, GA is especially advantageous for such high-dimensional problems. In practice,

these problems may very possibly appear since a large complex system is often composed of many small subsystem with little correlation. This intuitively explains the success of GA.

One major problem of genetic algorithm is that it is very difficult to perform the automatic identification and recombination of building blocks. In GA, *mutation* is used to find new building blocks. The mutation process is basically like a random sampling, but in *the building block space* and is very inefficient. And *crossover* is used to combine building blocks. However, it often either breaks existing building blocks or cannot combine them effectively[59]. In addition, its efficiency may also be reduced by hitchhiking effect[60], i.e, once a high-quality building block is found, it may quickly spread all over the population and suppress the finding of new building blocks. Mitchell[60] has found that even for a simple "Royal Road" function[61] specifically designed to suit the exploitation techniques of GA, simple multi-start hillclimbing can outperform genetic algorithm because of the reasons described above. Therefore, for the problems where mutation and crossover cannot effectively find and combine building blocks or the problems where "building block hypothesis" does not hold, GA will not perform as desired.

3.2.4 Simulated Annealing

Simulated annealing(SA)[13, 62, 8] is an optimization algorithm which mimics the physical annealing process, where a solid material is first melt by heating to a very high temperature and then cooled down at a very slow speed, the final product will settle in a highly ordered, crystalline state with the lowest energy. The basic search process of simulated annealing is similar to hillclimbing, however, it introduces an acceptance probability p for bad moves to avoid getting trapped in local optima. In each iteration, the search will select one point from the current neighborhood, if it is not as good as the current point, instead of refusing the new point like in hillclimbing, the search will accept it with a certain probability:

$$p = e^{|\Delta y|/T} \tag{3.3}$$

where T is termed as *temperature*, a parameter of SA algorithm, and Δy is the function value difference between the two points. Therefore, the acceptance probability is small for low temperature and very bad moves. In SA, the temperature T usually starts with a very high value, and the search will basically accept all moves and explore the parameter space with a way of random walk. With the search proceeding, the temperature is gradually lowered according to a *cooling scheme* and consequently the acceptance probability of bad moves is decreased. At each level of temperature, the algorithm executes a number of iteration until a certain *equilibrium* condition is satisfied. Finally when the temperature tends to zero, the search will only accept good moves and act basically like a hillclimbing procedure.

The exploration process of simulated annealing is random walk, and its exploitation process is hillclimbing. Different from the other algorithms, SA keeps adjusting the balance between exploration and exploitation during the search. The adjustment strategy is decided by the cooling scheme. In the beginning, *temperature* is high, it is more possible for random walk, that is, exploration is preferred than the exploitation. With *temperature* "cooling" down, random walk become less possible, that is, exploitation becomes increasingly preferred.

Simulated annealing has been used in various combinatorial optimization problems and is particularly successful in circuit design problems. It has been demonstrated[8] that with a high enough temperature, simulate annealing can converge in probability to the optimal solution. However, it should be noted that this is accomplished, at expense of efficiency, through extensive exploration at high temperatures and achieving equilibrium at each temperature level. The convergence speed of the algorithm is heavily dependent on the cooling scheme, and various types of cooling schemes have been proposed for speeding up convergence[63, 64]. Basically, SA is an algorithm more suitable for full optimization problems.

3.2.5 Tabu Search

Basic tabu search[65, 66, 67] is just a local search with memory. *Memory* is the most important concept in tabu search. Different types of memory-constructing schemes have lead to many variants of tabu search[68]. Compared with tabu search,

the other algorithms can be called memoryless methods. The basic idea of tabu search is to memorize the regions already visited in a tabu list, and prevent the search from revisiting these regions. when hitting a local optimum, tabu technique forces the search to accept bad moves. In this way, tabu search can escape the visited region of attraction and explore other parts of parameter space.

The exploitation in tabu search is hillclimbing, however, the exploration is much different from all other stochastic search algorithms since it actually explores the search space in a somewhat systematic way. Like multi-start algorithms, basic tabu search also emphasizes exploitation over exploration and is also based local search methods. Therefore, it has similar advantages and disadvantages to multi-start algorithms.

Tabu technique can be used with any other search algorithm to enhance its efficiency, in this sense, it is actually a meta-heuristic algorithm. Note that the speedup introduced by tabu technique is at the cost of more memory and higher complexity. Therefore, there exists a compromise between efficiency and memory.

3.3 No Free Lunch Theorem

Based on the investigation presented previously, we can summarize the features of various stochastic optimization algorithms in Table 3.1. As shown in the table, every stochastic has its advantages and disadvantages. With so many search algorithms, a question naturally arises: does there exist any all-purpose algorithm which can consistently outperform the other algorithms for every class optimization problems? Intuitively, advanced search algorithms, such as, GA or SA, should always perform better than random sampling. However, surprisingly, this intuition is wrong. It has been demonstrated in *No Free Lunch Theorem*[12, 11] that, no matter what performance metric is used, the average performance of any search algorithm is the same over all possible problems. In other words, no single algorithm can consistently perform efficiently in every class of problems. It might look discouraging that GA or SA is only as good as random sampling. However, it should be noted that NFL theorem holds only for the average performance over all possible problems which also include intractable problems, such as pure random objective functions.

Algorithm Names	Multi-start	Controlled Random Search	Genetic Algorithm	Simulated Annealing	Tabu Search
Exploration	Random Sampling	Random Sampling	Mutation	Random Walk	Tabu-based hill-climbing
Exploitation	Local Search	Downhill Simplex	Crossover	Hillcllimbing	Hillclimbing
Balance Strategy	Alternate exploration and exploitation and strong preference towards exploitation.	Start with exploration, then switch to exploitation only.	Alternately perform exploitation and exploration, well-maintained balance.	Stochastically select exploration or exploitation to execute, and the balance is controlled by the cooling scheme. Start with exploration, then increasingly shift the preference to exploitation during the search.	Alternate exploration and exploitation, strong preference to exploitation.
Advantages	Simple and effective in many applications.	Simple and demonstrated to be effective in practice.	Based on "Building Block Hypothesis", and suitable for the objective functions with the separable parameter structure.	Demonstrated to be effective for many combinatorial optimization problems, suitable for full optimization	Tabu techniques can be used with any algorithm to improve efficiency by avoiding revisits.
Disadvantages	Local search is sensitive to the effect of noises. Multi-start strategy may waste much time on trivial hills.	Convergence cannot be guaranteed. Perform like low-efficiency random sampling in the beginning stage.	Difficult to identify and combine "building blocks". Often need local search methods to improve efficiency, which is sensitive to noise.	Slow convergence and low efficiency. Only suitable for full optimization.	Hillclimbing is sensitive to noise. Tabu-based hillclimbing is not efficient to explore the parameter space.

Table 3.1: Comparison of Stochastic Optimization Algorithms

For one specific class of problems a search algorithm can perform more efficiently than the others. Therefore, the significance of NFL lies in that there is no general efficient search algorithm and a search algorithm has to take advantage of as much problem-specific knowledge as possible to achieve high efficiency in the given problem. However, such an algorithm may not perform well in other class of problems. Therefore, these exists a tradeoff between applicability and efficiency. In practice, a good search algorithm should find an appropriate balance between them.

CHAPTER 4
Design Issues of Efficient Search Algorithm

Solving an optimization problem always start with understanding the properties and requirements of the problem. This chapter will examine the main issues present in network optimization.

4.1 Design Requirements for Network Optimization

The main requirements for the desired search algorithm in network optimization are high efficiency, scalability to high-dimensional problem and robustness to noises in objective functions.

Efficiency is the most important factor which needs to be considered in the algorithm. Since network conditions keep changing all the time, a very efficient algorithm is required to quickly find better network configurations before significant changes happen to current network conditions. Accordingly, the emphasis of the algorithm should not be on seeking the optimum setting, but finding a better operating point within the limited time frame. This makes our desired algorithm different from traditional ones, such as genetic algorithm and simulated annealing, whose major objective is to find the global optimum. The high efficiency requirement is also due to the fact that the simulation of a complex network is often very time-consuming and it is necessary to reduce the number of function evaluations as much as possible.

Scalability to high-dimensional problems is another requirement for the desired search algorithm. Since a network often has hundreds of protocol parameters to be tuned, the desired algorithm should be able to handle a very large parameter space. For example, if a network comprises 100 nodes and each node is installed with at least one RED queue, which has four tunable parameters. If we want to optimize the parameter setting of all RED queues in this network, we will have an optimization problem with more than 400

parameters. Due to "curse of dimensionality", high-dimensional problems are usually much more difficult to solve than low-dimensional ones[6].

Robustness to noises in the objective function is another aspect needed to be considered because network simulation only provides us with approximate estimation of network performance, which means the objective functions in our problems are superimposed with small random noises due to inaccuracy in network modeling, simulation, etc.

In fact, the issues described above are also present in many practical problems[40, 69], therefore, the solutions to these issues are of more general significance. The following sections will investigate these issues in detail.

4.2 Distribution Function of Objective Function

For a black-box optimization problem, the features of its objective function is hard to characterized. One possible way is to use the distribution function of the objective function. This section will introduce its concept and major properties. The concept will be used in latter sections to investigate the properties of the objective function.

Given a measurable objective function $f : R^n \to R$ and a parameter space D, the distribution function of objective function value $y = f(\mathbf{x})$ is defined as:

$$\phi_D(y) = \frac{m(\{\, \mathbf{x} \in D \mid f(\mathbf{x}) \leq y \,\})}{m(D)}, \tag{4.1}$$

where $m(\cdot)$ is *Lebesgue measure*, a measure of the size of a set. For example, *Lebesgue measure* is just area on a R^2 space, and volume on a R^3 space, and so on. When given a specific y, this equation represents the ratio of the size of the set of points whose function values are below y to the whole parameter space. For example, in the objective function shown in Figure 4.1, the set of points below y consists of two separate regions A_1 and A_2. $\phi_D(y)$ is equal to the ratio of the size of A_1 and A_2 to the parameter space D.

The distribution function can also be understood from another point of view: if $\mathbf{x}$ is a random variable uniformly distributed on the parameter space D, then

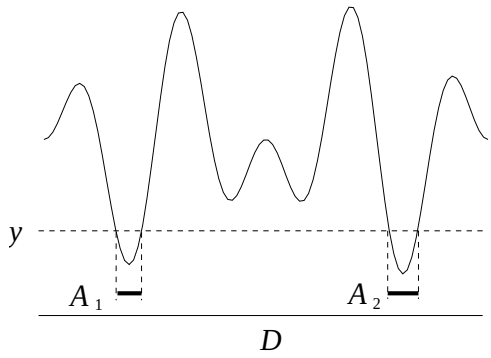


Figure 4.1: Illustration of obtaining $\phi_D(y)$

$y = f(\mathbf{x})$ will also be a random variable, whose distribution follows the one defined by Equation 4.1. According to this understanding, when $f(\mathbf{x})$ is not too complex, we can mathematically obtain its distribution function if we know the analytical expression of $f(\mathbf{x})$. Otherwise, we can only obtain the empirical distribution function of $f(\mathbf{x})$ by randomly sampling the parameter space. Since mathematically, $\phi_D(y)$ can be derived from $f(\mathbf{x})$, studying $\phi_D(y)$ can help us understand the properties of the objective function. In addition, random sampling is used in many stochastic search algorithms and may be the only choice for exploration when little *a priori* knowledge is available for the underlying problem. Therefore, it is also important to study the properties of the objective function under random sampling, i.e., $\phi_D(y)$.

Given a measurable function $f(\mathbf{x})$ and its value range $[y_{min}, y_{max}]$, its $\phi_D(y)$ 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 $m(x_*)/m(S)$ where x_* is the set of global optima. Without loss of generality, we assume that $f(\mathbf{x})$ is a continuous function and $m(z \in S | f(z) = y) = 0, \forall y \in [y_{min}, y_{max}]$, then $\phi(y)$ will be a monotonously increasing continuous function with a range of $[0, 1]$. An example of distribution function is shown in Figure 4.2.

4.3 Efficiency of Search Algorithms

There are two aspects for the performance of a search algorithm: effectiveness and efficiency. Effectiveness of a search algorithm means that the search algorithm should converge to the global optimum when it proceeds long enough. Efficiency

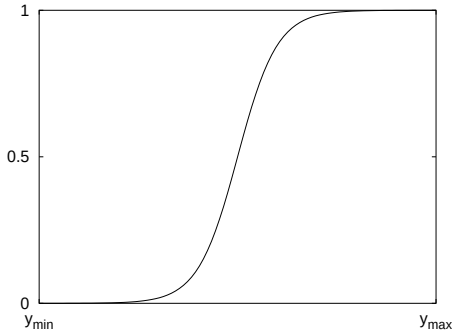


Figure 4.2: An example distribution function

means that the search algorithm should find the global optimum within the shortest time. Usually the efficiency is the most concerned perform metric for a search algorithm since the effectiveness is easy to achieve: simple random sampling can find the global optimum as long as enough time is given. This is especially true for on-line network optimization, where the efficiency is of utmost importance to its success.

4.3.1 Efficiency Measures of Optimization Algorithms

Although the efficiency of search algorithms is one of the most important concerns in optimization research, there has been no standard metric to directly measure the efficiency of a search algorithm. This is partly due to the lack of analytical understanding for search algorithms. This is also because that the performance of a search algorithm may vary with different classes of problems. In other words, the performance is dependent on the problem class. It is hard to find a general performance formula which count for all problem classes.

Since there is no absolute efficiency metric, two algorithms cannot be compared directly. In practice, search algorithms are empirically compared with experimenting on a certain benchmark objective function. Assume that the goodness of the optimization result can be measured by a metric M, the following two metrics are often used to assess the efficiency of a search algorithm:

- Given a certain goodness objective M_0 for the optimization result, the time T needed to obtain such a solution

- Given a certain time objective T_0, the goodness M of the optimization result obtained after a search process of time T_0

Since the time consumption of a search algorithm is greatly dependent on its specific implementation and the speed of the hardware on which tests are executed, it is hard to compare the performances of search algorithm by measuring their time consumption. Usually the evaluation of the objective function is considered the major task which takes up most of the computation time in an optimization process, therefore, the number of function evaluations is often used in place of the time consumption in performance comparison.

The relationship between the above two efficiency metrics can be illustrated in figure 4.3, where the optimization results of two algorithms are drawn as a function of the number of function evaluations. This type of function is called *convergence curve*, which better describes the performance of a search algorithm in a certain problem than a single number. The analytical expression of the convergence curve is usually very difficult to calculate. In practice, the curve can be empirically obtained with the average results of the tests on the underlying problems. From the convergence

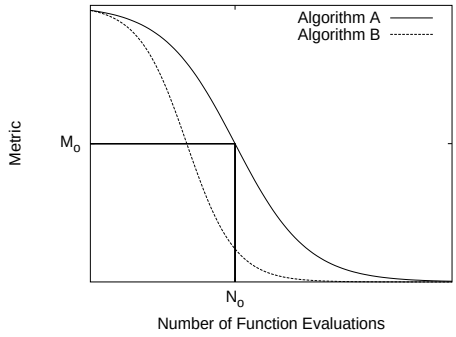


Figure 4.3: Relation between two performance metrics

curves shown in the figure, we can see that there is no essential difference between two metric. If algorithm A is better than B in terms of one metric, the same conclusion can be made by using the other metric. Which metric to choose is dependent on the objective of optimization. That is, whether the global optimal solution is desired or an approximate solution within a certain time limit can do.

Figure 4.3 shows an example where algorithm A is always more efficient than B. However, it is usually not the case in reality. The two performance metrics are both defined with a certain optimization objective, either in terms of the goodness of the solution or the time limit. The definition of the optimization objective may greatly affect the comparison of algorithm efficiency. Figure 4.4 shows such an example. If the second type of metric is used, i.e., the time limit is specified,

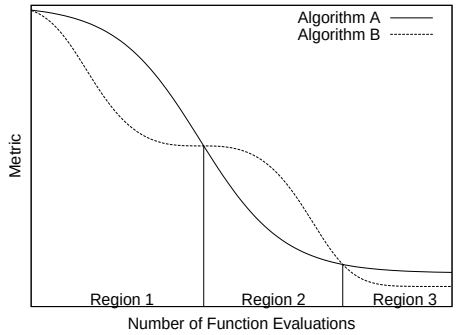


Figure 4.4: Inconsistency in efficiency comparison of search algorithms

the performance comparison shown in the figure can be divided into three regions with different time limit definitions. In the region 1, algorithm B performs better than A. However, when the objective is changed to region 2, algorithm A becomes better. In region 3, algorithm B outperform A again! In addition, for different optimization problem, the convergence curves of a search algorithm may change completely. Therefore, the efficiency comparison of search algorithms is dependent on two factors:

- the optimization objective
- the underlying problem (objective function)

From the above discussion, we can see that the efficiency comparison of search algorithm is a somewhat subjective measurement. It is greatly dependent on the optimization objective as well as the underlying problem and may vary with different objectives even in the same problem. This is why there is no consistent reports on the efficiency of search algorithms. Therefore, for a practical problem, the selection

of the appropriate search algorithm should take into account both the optimization objective and the properties of the problem.

4.3.2 Goodness Metric of Optimization Results

As mentioned in the previous section, a certain goodness metric of the optimization result has to be defined to compare the efficiency of search algorithms. Without loss of generality, assume that there exists a single optimum point $\mathbf{x}_*$ in the parameter space of the objective function. Let $\mathbf{x}$ denote a sample point in the parameter space, the following three metrics can be used to measure the goodness of $\mathbf{x}$:

- The metric measuring the distance of a sample to the global optimum is defined as:

$$M_x(\mathbf{x}) = \|\mathbf{x} - \mathbf{x}_*\| \tag{4.2}$$

  This metric seems reasonable and has been used in many optimization literature. However, there exists a serious problem for this metric. Search algorithms decide the goodness of a sample by its function value and return the sample with the best function value as the optimization result. However, M_x judges the result in a different way than search algorithms, i.e., by the distance to $\mathbf{x}_*$. This inconsistent may lead to a wrong determination on the efficiency of search algorithms. For example, for the objective function shown Figure 4.5, the second-best optimum $\mathbf{x}'$ is far away from the global optimum $\mathbf{x}_*$. Suppose a certain goodness objective M_x is defined and some points satisfying M_x has worse function values than $\mathbf{x}'$. If an algorithm is very quick to find $\mathbf{x}'$, then even if it can later reach some points in M_x, it will still keep $\mathbf{x}'$ as its optimization result until a point in M_x which is better than $\mathbf{x}'$ is found. Thus, the efficiency of this algorithm may be mistakenly measured if considering the time needed to obtain a solution satisfying M_x. To use M_x as the goodness metric, the search algorithm is required to use the same decision rule for the sample goodness, i.e, the distance to the global optimum. Unfortunately, this is not feasible since in practice the location of the global optimum is not known and is just what an optimization algorithm tries to find out.

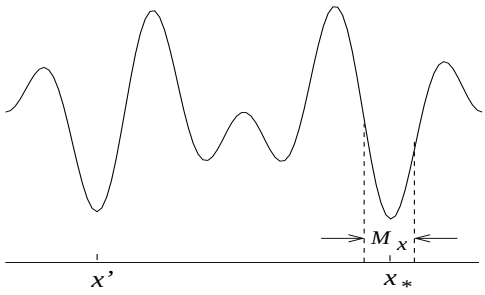


Figure 4.5: An example of M_x metric inconsistence

- The metric measuring the function value difference of a sample to the global optimum is defined as:

$$M_f(\mathbf{x}) = f(\mathbf{x}) - f(\mathbf{x}_*) \tag{4.3}$$

  This metric is consistent with the goodness decision rule of search algorithms and makes more sense in practice since most of time all we care about is the function value of the optimization result. Even if a point is far apart from the global optimum, we usually accept it as a near-optimal as long as its function value is good enough. In practice, since we don't know the value of $f(\mathbf{x}_*)$, we cannot compute the above metric. However, the relative metric of two samples can be computed by: $M_l(\mathbf{x}_1) - M_l(\mathbf{x}_2) = f(\mathbf{x}_1) - f(\mathbf{x}_2)$. Consequently, this metric can be used to compare the efficiency of search algorithms for a certain objective function without knowing its global optimum.

- The third metric is defined as:

$$M_\phi(\mathbf{x}) = \phi(f(\mathbf{x})) \tag{4.4}$$

  where $\phi(y)$ is the distribution function of $f(\mathbf{x})$ as defined in Equation 4.1. Note that for any measurable function, $\phi(y)$ is a monotonous increasing function of y, its maximum value is 1 when $y = y_{max}$ and its minimum value is achieved at the optimal function value. Therefore, it actually reflects the same performance measurement as M_f. M_f and M_ϕ can be related through the distribution function of the objective function as shown in Figure 4.6. This

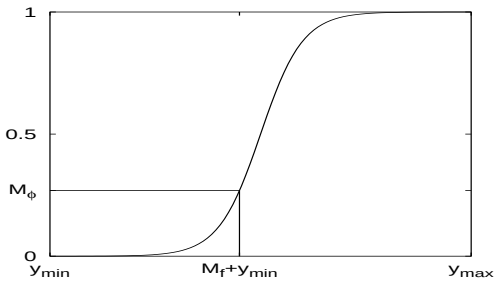


Figure 4.6: An example distribution function showing the correlation between M_f and M_ϕ

metric can not be used for the empirical efficiency study of search algorithm since it requires the computation of $m(z \in D|f(z) \leq y)$, which is usually intractable. However, in many cases, it is possible to analytically compute the value of $\phi(y)$, thereby, $M_\phi(\mathbf{x})$, with help of some statistical techniques without the need for any knowledge of $f(\mathbf{x})$, For example, we can easily calculate that the average M_ϕ metric of a random sampling after n iterations will be $\frac{1}{n+1}$[70, p.14]. Therefore, with metric, it is possible to analytically compare the performances of some algorithms.

M_x or M_f are the goodness measures of absolute type, for which the same value may means totally different in different problems. On the contrary, M_ϕ is a relative measure, for which the same value usually has the same meaning for different problems. Therefore, when comparing the performance of a search algorithm in different problems, M_ϕ is more appropriate.

4.3.3 Comparison of Some Algorithms with M_ϕ

Based on the metric M_ϕ, this section compares the performance of random sampling and multi-start algorithms.

Random sampling returns the lower extreme of random samples as the optimization result. According to order statistic theory, the average result metric of random sampling after n random samples can be obtained as[70, p.14]:

$$C_r(n) = \frac{1}{n+1} \tag{4.5}$$

Multi-start algorithms use random sampling for exploration and local search methods for exploitation. For a multi-start algorithm, the efficiency is achieved with its local search method, which basically changes the distribution function of the underlying objective function to a simpler one. To illustrate this point, Figure 4.7 shows an example objective function $f(x)$. After applying a local search for each random sample, the objective function will looks like $g(x)$ for the exploration part, i.e., random sampling, of the multi-start algorithm. And Figure 4.8 shows the distribution functions $d_f(y)$ and $d_g(y)$ for $f(x)$ and $g(x)$, respectively. Note that $d_f(y) < d_g(y)$ always holds. This is because $g(x) < f(x)$, therefore, the set $\{z \in D | g(z) \leq y\}$ is always larger than $\{z \in D | f(z) \leq y\}$.

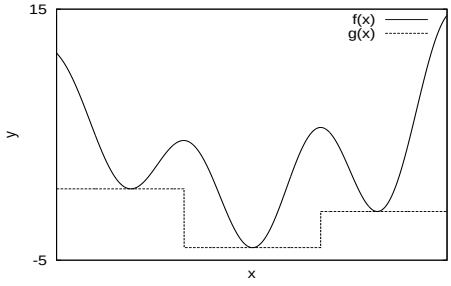


Figure 4.7: Objective function change in multi-start algorithms

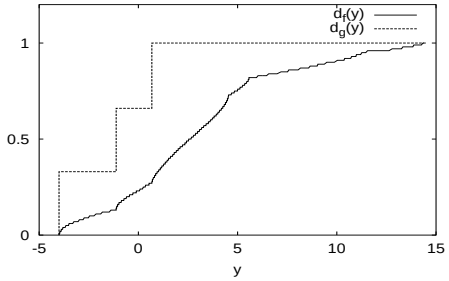


Figure 4.8: Distribution function change in multi-start algorithms

If a function $h : [0, 1] \rightarrow [0, 1]$ can be defined, which maps the new distribution function to the old one, i.e., $h(x) = d_f(d_g^{-1}(x))$, this function will determine how efficient the multi-start algorithm will be. Figure 4.9 shows an example mapping function for the objective function in Figure 4.7. Note that $h(x) < x$ always holds since $d_f(y) < d_g(y)$.

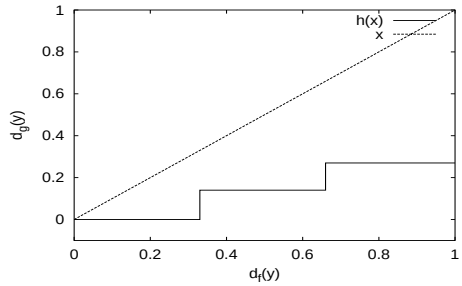


Figure 4.9: Example mapping function from $d_g(y)$ to $d_f(y)$

Assume that the local search procedure takes an average of l function evaluations to reach a local optimum, then the average result metric of the multi-start algorithm after n samples can be approximately calculated as:

$$C_m(n) = h\left(\frac{1}{\lfloor n/l \rfloor + 1}\right), \tag{4.6}$$

where $\lfloor \cdot \rfloor$ takes the integer floor of a decimal number. Therefore, for a multi-start algorithm to be better than random sampling, the following inequality has to hold:

$$C_m(n) - C_r(n) = h\left(\frac{1}{\lfloor n/l \rfloor + 1}\right) - \frac{1}{n+1} < 0 \tag{4.7}$$

As shown in the above equation, the performance of multi-start algorithm suffers from the term $\lfloor n/l \rfloor$. However, it takes advantage of the property $h(x) < x$ to make up this loss of efficiency and outperform random sampling. Therefore, the more far below $h(x)$ is to x, the better the performance of the multi-start algorithm will be. Otherwise, if $h(x)$ is close to x, the performance of the multi-start algorithm may become worse than random sampling. When $n \gg l$, i.e., the search is executed for a large number of function evaluations, $\frac{1}{n+1} \approx \frac{1}{\lfloor n/l \rfloor + 1}$. In this case, multi-start algorithms will perform better since $h(x) < x$ always holds.

4.4 Large-scale Optimization Problems

One of the important goals in network optimization is to handle large-scale problems which has hundreds or thousands of parameters. Therefore, it is important

to investigate the properties of large-scale problems. "Curse of dimensionality" is present in large-scale problems[6] and most search algorithms cannot scale well with increasing dimension. This section attempts to explain why large-scale problems are difficult. Specifically, the questions we want to answer are: In which ways are large-scale problems different from small-scale ones? Why will these changes happen? How will these changes affect the performance of search algorithms?

As the dimension of the problem increases, the following features will be present:

The size of the parameter space increase exponentially because of "curse of dimensionality". For example, if the range of every parameter is large than a, then the size of the parameter space $m(D)$ increases in a way of a^n.

The number of local optima may increase exponentially in many cases. Many paper have commented that most practical problems tend to have a large number of local optima[34, 69]. For example, for the molecular conformation problem (finding the structure or relative positions of a cluster of atoms that minimize the potential energy of the structure), Hoare[15] has conjectured that the general number of local minimizers for an N-atom cluster using Lennard-Jones potential energy function is $O(e^{N^2})$. This is also been confirmed by so-called "Levinthal Paradox"[16].

Central Limit Catastrophe may occur in many cases, i.e., local optima in large-scale problems tend to all have average quality with little variance and are significantly worse than the global optimal solution. In other words, when the problem size increases, an increasing percentage of the local optima will be clustered near the average value which is far from the global optima. This phenomena has been found in many practical problems, such as Traveling Salesman Problem(TSP), graph bisection and molecular conformation problems[34, 71, 72].

"Globally convex" or "big valley" structure is often present in many practical problems[73, 50]. That is, high-quality local optima tend to center around the global optimum with small average distance, whereas low-quality local optima

tend to distribute around the global optimum with larger average distance. Boese[71] has demonstrated the existence of this structure in complex TSP and graph bisection problems and presented an intuitive graph for this structure as shown in Figure 4.10. The same structures have also been found in circuit/graph partitioning and job-shop scheduling, etc[34]. Leary[74] also confirmed that there exist similar "funnel" structures in molecular conformation problems.

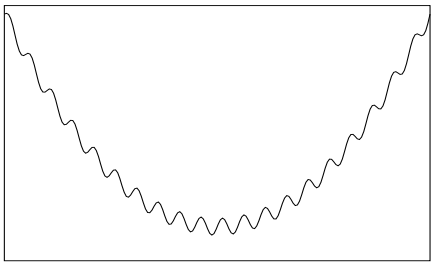


Figure 4.10: Big valley structure

One possible reason for these phenomena the so-called parameter separability property in many large-scale problems, i.e., the objective function can be represented by

$$f(\mathbf{x}) = \sum_{i=1}^{m} f_i(\mathbf{x}_i) \tag{4.8}$$

where $\mathbf{x}_i$ are the mutual exclusive subsets of elements in $\mathbf{x}$. This property often appears in practical large-scale problems since a large-scale system usually comprises many smaller subsystem which are not or weakly dependent on each other. This is consistent with the "Building Block Hypothesis" which genetic algorithms exploit to improve efficiency. The wide success of genetic algorithms also indirectly proves the existence of the separability property in practical problems. Many widely used benchmark functions, such as Rastrigin and Schewefel, also have this structure. For a objective function with parameter separability, we can know from Central Limit Theorem that its distribution function will follow normal distribution. Furthermore, if we assume m in Equation 4.8 increases with n, then most part of the parameter space will have the average function value with small deviation(relative to the av-

erage), i.e., Central Limit Catastrophe occurs. For example, the empirical density functions(derivative of distribution function) for Rastrigin benchmark function in different dimensions are shown in Figure 4.11. Note the function value in the figure has been normalized to show the relative deviation. The parameter separability

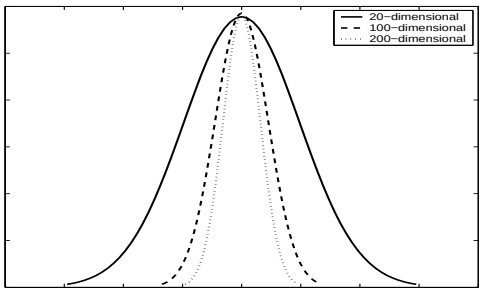


Figure 4.11: Density function of Rastrigin benchmark function in different dimensions

property can also explain the phenomenon that the number of local optima increases exponentially with dimension since the number of local optima in $f(\mathbf{x})$ is equal to the product of the local optima number for each $f_i(\mathbf{x}_i)$.

According to the above discussion, as the dimension of a problem increases, the properties of the underlying objective function also change. Therefore, when studying the scalability of a search algorithm, we actually examine its performance on different problems. As a result, a search algorithm good at low dimensions may not be good at high dimensions since the properties which they aim to exploit have diminished. If the changes described as above appear in a practical problem, only those algorithm which are not sensitive to these changes can scale well with the increase of dimension. In other words, only the algorithms which exploit the properties not affected by these changes can perform well with increasing dimension.

Random sampling does not exploit any problem-specific information, therefore, it should exhibit the same performance across all class of problems according to the above discussion. However, it is usually believed that the performance of random sampling degrades exponentially with the problem dimensionality. The contradiction is due to the metrics used in the efficiency study for different problem dimensions. In the study, M_f or M_x is used to measure the goodness of an

optimization result. However, as described before, the same M_f or M_x may means different things in different problems and M_ϕ is more appropriate in these cases. For example, if the same optimization objective M_x is defined for different problem dimensions, this actually implies an exponentially increasing optimization objective in terms of M_ϕ in higher dimensions since the size of the parameter space increases exponentially with dimensionality. Implicit increasing optimization objective by use of M_x or M_f is another factor which make many algorithms not scale well to high-dimensional problems. If the metric M_ϕ is directly used in the efficiency study, this problem will not appear. In fact, for any dimension, random sampling has the same performance in terms of M_ϕ.

CHAPTER 5
Recursive Random Search Algorithm

Based on the previous investigation of the problems in network optimization, this chapter presents a Recursive Random Search(RRS) algorithm which is designed to perform efficient optimization in the concerned problems. The most remarkable feature of this algorithm is that it is completely based on random sampling. Random sampling is usually believed to be lacking in efficiency. However, as we describe in the following, it is actually very efficient in its initial steps. The RRS algorithm exploits this feature and maintains the high efficiency by constantly restarting random sampling in new sample spaces. Besides its high efficiency, the RRS algorithm has additional advantages. First, in the situation where the evaluation of objective function is affected by noises, the RRS algorithm is more robust than the algorithms using local search methods. In addition, the RRS algorithm is especially efficient for the objective function with negligible parameters.

In the following, the design ideas of the RRS algorithm is first described and the details of the algorithm is then presented. Finally, the advantages described above are validated with the tests on a suite of benchmark functions.

5.1 Design Ideas of Recursive Random Search

In this section, we first describe the initial high-efficiency property of random sampling, which is the base of the Recursive Random Search algorithm and then present the strategy of RRS for the balance between exploration and exploitation.

5.1.1 Efficiency of Random Sampling

Random sampling may be the simplest and most widely used search technique, which takes random samples from a uniform distribution over the parameter space. This technique provides a strong probabilistic convergence guarantee but is generally considered to be lacking in efficiency. Opposite to this common belief, we will show in the following that random sampling is in fact very efficient in its initial steps and

its inefficiency is from the latter sampling.

Assume an objective function $f(\mathbf{x})$ is defined on the parameter space D and its value range is $[y_{min}, y_{max}]$. Given a number $y_r \in [y_{min}, y_{max}]$ such that $\phi_D(y_r) = r, \quad r \in [0, 1]$, we can define a *r-percentile* set in the parameter space D:

$$A_D(r) = \{\, \mathbf{x} \in D \mid f(\mathbf{x}) \leq y_r \,\} \tag{5.1}$$

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:

$$P(\mathbf{x}^n_{(1)} \in A_D(r)) = 1 - (1 - r)^n = p \tag{5.2}$$

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

$$r = 1 - (1 - p)^{1/n} \tag{5.3}$$

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 5.1 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 5.1, 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.

To maintain its initial efficiency, random sampling has to be "restarted" when 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

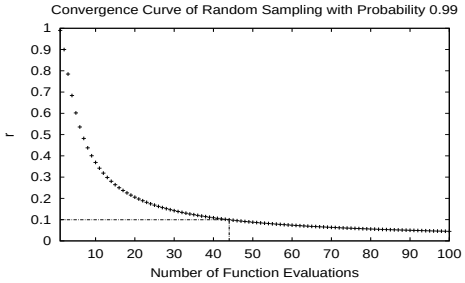


Figure 5.1: Convergence curve of random sampling with probability 0.99

random sampling in the new sample space. This restart strategy is used as the base of Recursive Random Search to maintain the high efficiency of random sampling.

5.1.2 Balance between Exploration and Exploitation

A stochastic search algorithm consists of two procedures: exploration and exploitation. Basically, the RRS algorithm uses random sampling for exploration and recursive random sampling for exploitation. The *balance* between exploration and exploitation is essential to the efficiency of an algorithm. Ideally it should only execute the exploitation procedure in prospective areas. However, it is difficult to determine which areas are more promising and should be exploited. Many algorithms, such as multi-start 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 5.2. The left graph shows a contour plot of a 2-dimensional multi-modal 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 for many optimization problems since the region of attraction containing the global optimum usually has the largest size [6]. 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.

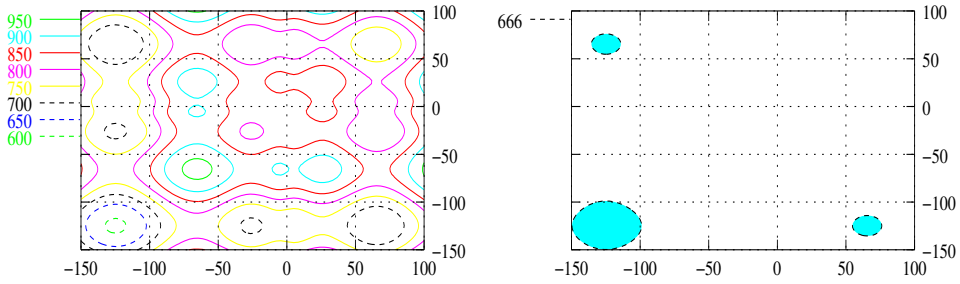


Figure 5.2: Contour plot of an objective function(left) and its region of $A_D(0.05)$(right)

The size of $A_D(r)$ region identified by exploration is desired to be 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.

5.2 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.

5.2.1 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 5.2): $n = \frac{\ln(1-p)}{\ln(1-r)}$

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-align 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

(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.

5.2.2 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 5.3 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 5.3) is large, that means most points in $N(\mathbf{x}_0)$ are better than $\mathbf{x}_0$. Therefore, if we do random

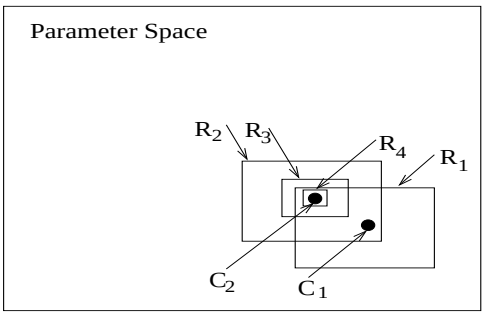


Figure 5.3: Shrink and Re-align Process

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 5.3, 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 5.3, 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.

5.2.3 Remarks on the RRS Algorithm

Recursive random search is an algorithm completely based on random sampling. It attempts to maintain the high efficiency of random sampling by constantly changing the sample space. The exploration process of RRS accomplishes two tasks. One is to filter out most of trivial areas in the parameter space and consequently the optimization can be focused on the areas which contain high-quality solutions. The other task is to provide a good starting point for the exploitation process. This is to make the most of the efficiency of the exploration phase and only execute exploitation when exploration cannot supply enough efficiency.

The ideal behavior of a search algorithm is to first inspect the macroscopic of the objective function, and then examine microscopic features in selected areas. The search process of RRS algorithm is fully consistent with this idea. In the beginning of the search, random sampling is performed over the whole parameter space and the overall structure of the objective function is examined. With the search continuing, the sample space gradually shrinks and consequently the search becomes increasingly focused on microscopic information until it finally converges to a local optimum. The search algorithms deviating the above ideal behavior usually cannot perform efficiently. For example, multi-start type algorithms start a local search from every random sample without considering macroscopic features, therefore, their performance often suffers greatly from the time waste on trivial areas.

Since the RRS algorithm performs the search process based on stochastic information, its performance will be less affected by noises imposed on the objective function than those using local search methods. In addition, random sampling is more efficient when dealing with an objective function with some negligible parameters, i.e., the parameters which contribute little to the function value. Since random samples can maintain its uniform distribution in the subspace composed of only

those important parameters, this effectively removes the negligible parameters from the optimization. In this way, the search efficiency can be greatly improved.

5.2.4 Efficiency of the RRS Algorithm

In this section, we will analyze the major reasons that the RRS algorithm can perform more efficiently than other algorithms. Black-box optimization problems are known to be NP-hard[75]. Because of the complexity of these problems, it is usually very difficult to quantitatively analyze the efficiency of an optimization algorithm. As indicated in [9], "the variety of techniques that have been proposed is impressive, but their relative merits have neither been analyzed in a systematic manner nor properly investigated by computational experiments on a wide range of problems." To study the efficiency of an optimization algorithm, the convergence curve expression, $m(n)$, has to be derived where n denotes the number of function evaluations and $m(n)$ is the performance metric of the algorithm after n function evaluations. Since the performance of an optimization algorithm varies with problems according to No Free Lunch Theorem, it is normally not possible to obtain a uniform formula of $m(n)$ such that the algorithms can be compared without reference to a specific problem. Even given a specific problem, it is often too difficult to analytically derive the expression of $m(n)$ because of the complexity of black-box optimization. It may be possible to derive a formula of $m(n)$ for a certain algorithm in a simple benchmark problem, however, too much simplification and restriction makes it useless in studying the performance of the algorithm in practice. This is the reason that there has been no analytical performance result for stochastic optimization algorithms, such as, genetic algorithm, simulated annealing and controlled random search, though these algorithms have been widely studied and used in practice.

The performance of stochastic algorithms is typically studied in a *empirical* way with benchmark tests like in the vast amount of optimization literature[76, 60, 10, 69, 77, 78, 72, 79]. The analytical work is mainly centered on the demonstration of the effectiveness of a stochastic algorithm, i.e., its convergence to the global optima in probability, such as the convergence of simulated annealing[13, 64].

However, for some stochastic algorithms, even this convergence property cannot be analytically obtained. For example, controlled random search(CRS) is a widely used black-box optimization algorithm[30, 56, 80, 40, 81, 9], however, as indicated in [57, 82], "a disturbing fact concerning CRS algorithms is their totally heuristic nature with no theoretical convergence properties" and "up to date there was no analysis on the performance of the algorithm". Even for genetic algorithm which has numerous successful applications in practice, "no theoretical convergence results for genetic algorithms are known."[83] The convergence of RRS is straightforward since it is based on random sampling and any stochastic algorithm including random sampling will converge to global optima in probability. In the following, we will only present a *qualitative* analysis for the efficiency of the RRS algorithm.

First of all, the balance strategy of RRS can filter out those trivial hills, i.e., only start exploitation in a certain r-percentile set. As we shown in the previous example in Figure 5.2, there are tens of hills in the original objective function, however, in RRS, the number of hills to be exploited is reduced to only three. Therefore, the RRS algorithm will perform much more efficient than multi-start local search methods for the objective function with many trivial hills. One example of such objective functions is "global convex" function, which appears in many practical optimization problems.

The efficiency of RRS also comes from its robustness to noises. The effect of noises is usually embodied as small random fluctuations imposed on the original objective function. For example, Figure 5.4 shows a 2-dimensional empirical objective function obtained with network simulation. It can be seen that there exist many irregular small fluctuations on the overall structure. For such objective function, multi-start local search algorithms will explore each of those small noise hills and hence basically perform like a random sampling. However, the RRS algorithm does not base its search strategy on a local structure, instead it starts with a large sample space and examines the overall structure first, and then increasingly adjusts the sample space to converge to a local optimum. Therefore, the RRS algorithm will perform more efficient in noise-affected objective functions as exemplified in Figure 5.4 than multi-start algorithms.

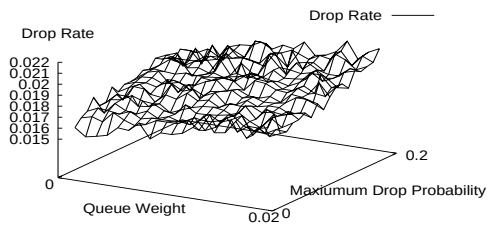


Figure 5.4: An empirical objective function obtained with network simulation

Furthermore, RRS can automatically exclude negligible parameters from the optimization while local search methods cannot. In practice, a problem often includes many parameters which are insignificant in terms of its contribution to the objective function. For these problems, RRS will perform much more efficiently than those based on local search methods.

5.3 Test Results

As mentioned before, the design objectives of Recursive Random Search are: high efficiency, scalability to high-dimensional problems and robustness to noises. This section will present the performance tests of the RRS algorithm in these aspects.

Usually, the benchmark tests of a search algorithm are performed by examining the number of function evaluations that it requires 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, the above method is not adopted in the tests presented in this section. Instead, we execute the search algorithm for a certain number of function evaluations, and draw the convergence curve of the optimization, i.e., the optimization result as a function of the number of function evaluations. The performance of the algorithms is studied based on these convergence curves.

5.3.1 Benchmark Functions

A suite of classical benchmark functions have been used in our performance tests. Most of them have a large number of local optima and are considered very difficult to optimize. We describe these function in the following. For these functions, we assume that the parameter $\mathbf{x}$ is a n-dimensional vector and x_i denotes its ith element.

1. The first function is a simple square sum function:

$$f(\mathbf{x}) = \sum_{i=1}^{n} (x_i^2) \tag{5.4}$$

where $-500 \leq x_i \leq 500, i = 1 \ldots n$. The global minimum is 0 at $x_i = 0, i = 1 \ldots n$. This function is often used for the first benchmark test of a search algorithm.

2. Rastrigin's Function was proposed by Rastrigin[84] and generalized as the following format by Mühelenbein, Schomisch and Born[85]

$$f(\mathbf{x}) = n \cdot A + \sum_{i=1}^{n} (x_i^2 - A \cdot \cos 2\pi x_i) \tag{5.5}$$

where $-5.12 \leq x_i \leq 5.12, i = 1 \ldots n$ and $A = 10$. The global minimum is 0 at $x_i = 0, i = 1 \ldots n$. There are a large number of local minima in this function, for example, grid points with $x_i = 0$ except one coordinate, where $x_j = 1.0$, give $f(x) = 1.0$. It is considered a very difficult benchmark function. A 2-dimensional Rastrigin function is shown in Fig 5.5.

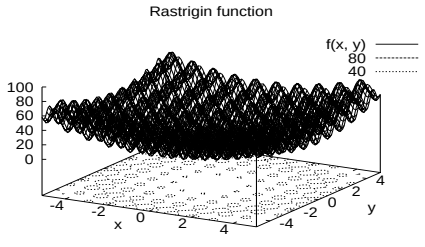


Figure 5.5: Rastrigin Function

3. Rosenbrock's Saddle[86] is defined as:

$$f(x) = \sum_{i=1}^{n-1}(100 \cdot (x_{i+1} - x_i)^2 + (1 - x_i)^2) \tag{5.6}$$

where $-2.048 \leq x_i \leq 2.048, i = 1 \ldots n$. The global minimum is 0 at $x_i = 1.0, i = 1 \ldots n$. This function has a long curved valley which is only slightly decreasing. There are strong interactions between variables. A 2-dimensional Rosenbrock function is shown in Fig 5.6.

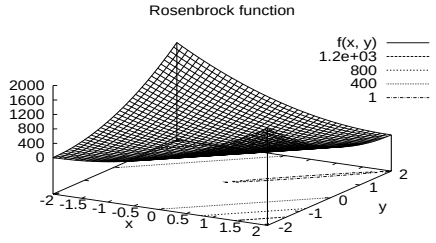


Figure 5.6: Rosenbrock's Saddle Function

4. Griewangk's Function is one of the most difficult global optimization test functions[6]:

$$f(\mathbf{x}) = 1 + \sum_{i=1}^{n}(x_i^2/4000) - \prod_{i=1}^{n}(\cos(x_i/\sqrt{i})) \tag{5.7}$$

where $-600 \leq x_i \leq 600, i = 1 \ldots n$. The global minimum is 0 at $x_i = 0, i = 1 \ldots n$. Typical n is 10, for which there are four local minima $f(x) \approx 0.0074$ at $x \approx (\pm\pi, \pm\pi \cdot \sqrt{2}, 0, ..., 0)$. This function has a product term, introducing an interdependency between the variables. This is intended to disrupt optimization techniques that work on one function variable at a time. A 2-dimensional Griewangk function is shown in Fig 5.7.

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

$$f_\sigma(\mathbf{x}) = 1 + \sigma \sum_{i=1}^{n}(x_i^2/4000) - \prod_{i=1}^{n}(\cos(x_i/\sqrt{i})) \tag{5.8}$$

This function is a bumpy quadratic when σ is one and a product of cosines

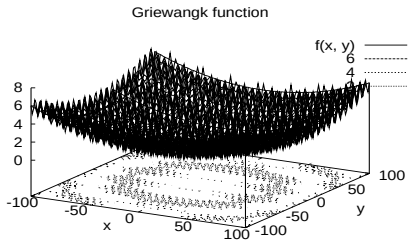


Figure 5.7: Griewangk's Function

when σ is zero. The value of σ decides the difference between the local minima. When σ approaches 0, the values of local minima become similar.

5. Ackley's Function[87] is defined as:

$$f(x) = 20 + e - 20 \cdot e^{-0.2 \cdot \sqrt{\frac{1}{n} \cdot \sum_{i=1}^{n} x_i^2}} - e^{\frac{1}{n} \cdot \sum_{i=1}^{n} \cos 2\pi x_i} \tag{5.9}$$

where $-30 \leq x_i \leq 30, i = 1 \ldots n$. The global minimum is 0 at $x_i = 0, i = 1 \ldots n$. A 2-dimensional Ackley function is shown in Fig 5.8.

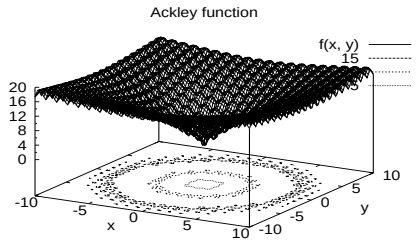


Figure 5.8: Ackley's Function

6. Shekel's family[6]

$$f(\mathbf{x}) = -\sum_{i=1}^{m} \frac{1}{(\mathbf{x} - a_i)^T (\mathbf{x} - a_i) + c_i)}$$

where $x_i \in [0, 10]$, $n = 4$, a_i is a 4-element vector. This function has m minima in positions a_i with levels c_i. The values of a_i and c_i are given in Table 5.1 for $m \leq 10$.

i	a_i	c_i
1	(4, 4, 4, 4)	0.1
2	(1, 1, 1, 1)	0.2
3	(8, 8, 8, 8)	0.2
4	(6, 6, 6, 6)	0.4
5	(3, 7, 3, 7)	0.4
6	(2, 9, 2, 9)	0.6
7	(5, 5, 3, 3)	0.3
8	(8, 1, 8, 1)	0.7
9	(6, 2, 6, 2)	0.5
10	(7, 3.6, 7, 3.6)	0.5

Table 5.1: Parameters for Shekel's function family

7. Hartman's family[6]

$$f(x) = -\sum_{i=1}^{m} c_i \exp(-\sum_{j=1}^{n} a_{ij}(x_j - p_{ij})^2)$$

where $x_i \in [0, 1]$, $m = 4$, $n = 3, 6$. This functions has m minima in positions p_i with levels c_i. a_i, c_i and p_i are given in Table 5.2 for Hartman3($n = 3$) and 5.3 for Hartman6 ($n = 6$).

i	a_i	c_i	p_i
1	(3, 10, 30)	1	(0.3689, 0.1170, 0.2673)
2	(0.1, 10, 35)	1.2	(0.4699, 0.4387, 0.7470)
3	(3, 10, 30)	3	(0.1091, 0.8732, 0.5547)
4	(0.1, 10, 35)	3.2	(0.03815, 0.5743, 0.8828)

Table 5.2: Parameters for Hartman's function family

i	a_i	c_i	p_i
1	(10, 3, 17, 3.5, 1.7, 8)	1	(0.1312, 0.1696, 0.5569, 0.0124, 0.8283, 0.5886)
2	(0.05, 10, 17, 0.1, 8, 14)	1.2	(0.2329, 0.4135, 0.8307, 0.3736, 0.1004, 0.9991)
3	(3, 3.5, 1.7, 10, 17, 8)	3	(0.2348, 0.1451, 0.3522, 0.2883, 0.3047, 0.6650)
4	(17, 8, 0.05, 10, 0.1, 14)	3.2	(0.4047, 0.8826, 0.8732, 0.5743, 0.1091, 0.0381)

Table 5.3: Parameters for Hartman's function family

8. Six hump camel back function[6]

$$f(x_1, x_2) = (4 - 2.1x_1^2 + x_1^4/3)x_1^2 + x_1x_2 + (-4 + 4x_2^2)x_2^2$$

where $x_1 \in [-3, 3], x_2 \in [-2, 2],\ f_* \approx -1.0316285$.

9. Goldstein Price[6]

$$(1 + (x_1 + x_2 + 1)^2(19 - 14x_1 + 3x_1^2 - 14x_2 + 6x_1x_2 + 3x_2^2))\cdot$$

$$(30 + (2x_1 - 3x_2)^2(18 - 32x_1 + 12x_1^2 + 48x_2 - 36x_1x_2 + 27x_2^2))$$

where $-2 \leq x_i \leq 2,\ f_* = 3$.

5.3.2 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 multi-start pattern search. Controlled random search is recommended for black-box optimization problems in many literature[9, 40]. Multi-start pattern search is chosen because multi-start type algorithms are always one of the most popular methods in practice[69] 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[51, 52, 53]. And pattern search[47] is one of direct search techniques which are usually recommended for black-box optimization problems[39].

In the tests, the search algorithms are executed on each function with the function dimension varying from 20 to 2000. To eliminate randomness caused by the 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. We have chosen $A = 10$ for Rastrigin function and 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 benchmark function 1-5 are shown in Fig 5.9-5.13. 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 5.12, for 200-dimensional Ackely function, the performances of RRS and the multi-start algorithm are close. However, the RRS tends to converge to the optimal solution quickly while the convergence curve of multi-start 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 multi-start pattern search. However, with the search continuing, its performance quickly degrades and falls far behind the other two algorithms.

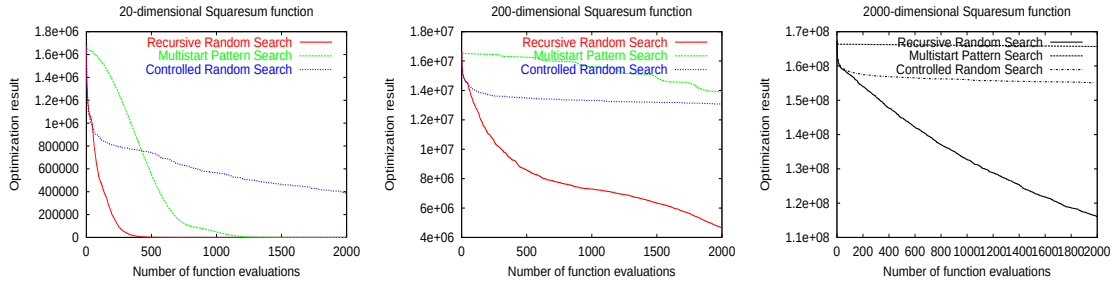


Figure 5.9: Performance tests on SquareSum function

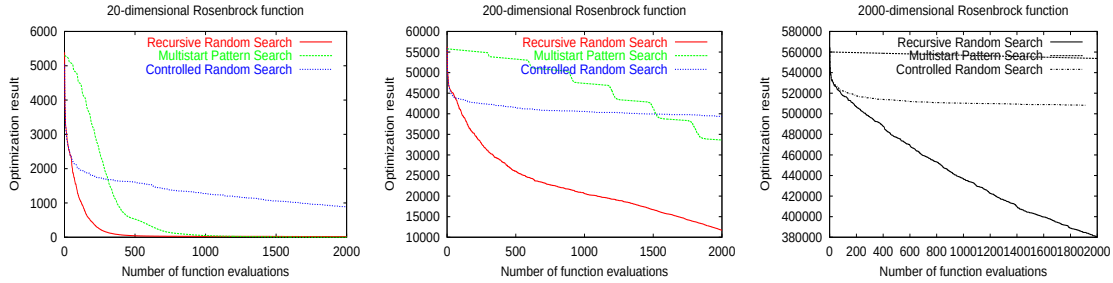


Figure 5.10: Performance tests on Rosenbrock function

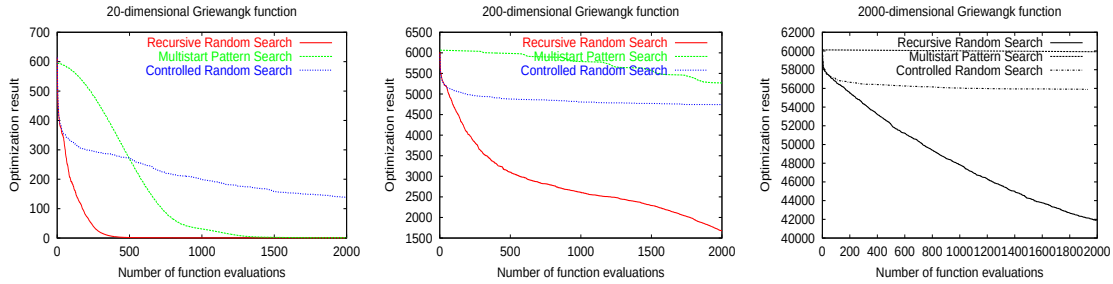


Figure 5.11: Performance tests on Griewangk function

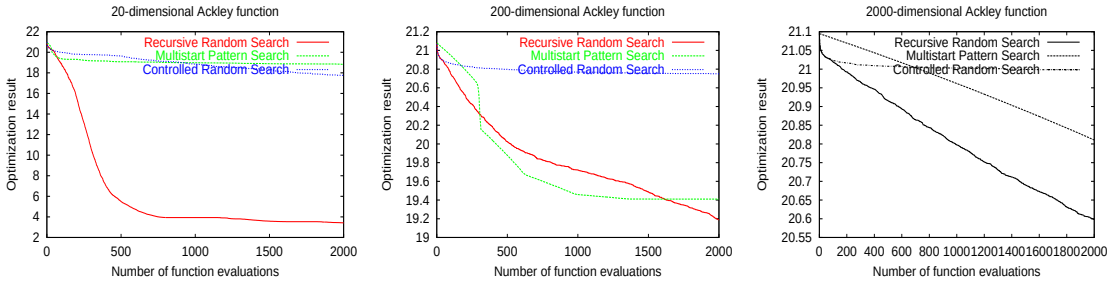


Figure 5.12: Performance tests on Ackley function

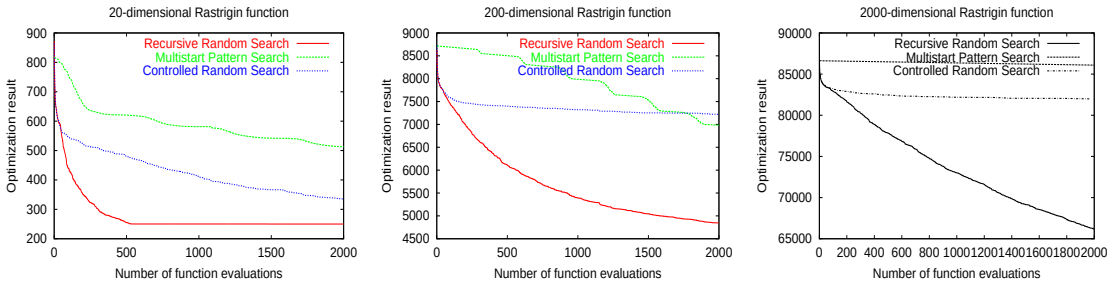


Figure 5.13: Performance tests on Rastrigin function

We also test the RRS algorithm on some low-dimensional classical benchmark functions (function 6-9 as listed before) 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 5.4 which shows the best-so-far function values after 75 function evaluations. It can be observed that the 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 5.4: Benchmark test results showing better optimization results found by RRS

5.3.3 Tests on Noise-Resistance of RRS

This section will compare the performance of RRS and multi-start 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. Recall that Rastrigin function is defined as:

$$f(\mathbf{x}) = n \cdot A + \sum_{i=1}^{n} (x_i^2 - A \cdot \cos(2\pi x_i)) \tag{5.10}$$

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 5.14 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 multi-start pattern search while the effect on RRS is slight.

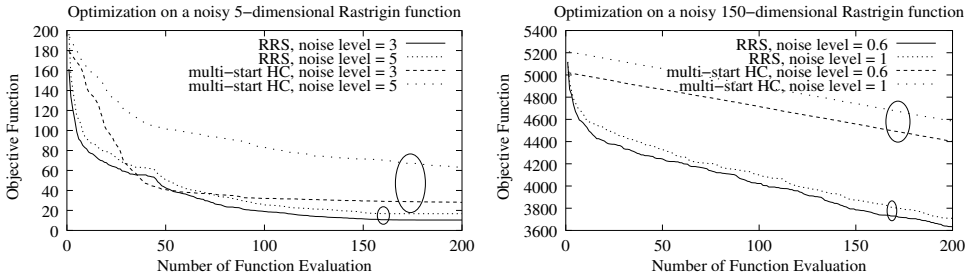


Figure 5.14: Noise-resistance tests of search algorithms

5.3.4 Tests on Objective Functions with Negligible Parameters

In many practical problems, it is common that only some of the parameters are important in terms of their contributions to the objective function and the

optimization result is largely affected by these parameters. In contrast, the other parameters are negligible in the sense that they have little effect on the value of the objective function. It is often difficult to identify these negligible parameters and exclude them from the optimization process. For such cases, the RRS algorithm is especially efficient since it automatically achieves the exclusion of trivial parameters with random sampling. The tests in this section will validate this property of RRS. To simulate the occasion with trivial parameters, an n-dimensional test function in Equation (5.11) is used:

$$f(\mathbf{x}) = \sum_{i=1}^{5} x_i^2 + 10^{-12} \cdot \sum_{i=5}^{n} x_i^2 \tag{5.11}$$

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 multi-start pattern search are compared. Figure 5.15 shows the test results. It can be seen that the introducing of trivial parameters can hardly affect the performance of the RRS algorithm while the performance of multi-start patter 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.

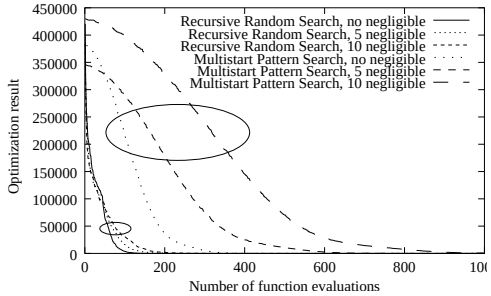


Figure 5.15: Performance tests on objective functions with negligible parameters

CHAPTER 6
Unified Search Framework

Despite the rapid progress in recent years, two major issues remain in black-box optimization, i.e., how to select an appropriate optimization algorithm for a give problem and how to manage and take full advantage of available computing resource. In this chapter, we propose a Unified Search Framework (USF) as a general solution to address these problems.

According to No Free Lunch theorem[12], no algorithm can consistently outperform the others for every class of problem. As a result, given an optimization problem, it is very important to choose an appropriate optimization algorithm which best exploits the structural properties of the concerned problem. Recursive Random Search is designed to be a general solution for typical network optimization problems with features described before. The design objective of RRS is to strike an appropriate balance between applicability and efficiency. In other words, RRS tries to maintain general applicability to typical network optimization problems while striving for high efficiency. For certain problems where there exist the extra exploitable structures, RRS can be outperformed by a search algorithm which aggressively exploits the problem-specific knowledge.

In light of the fact that one single optimization technique is not sufficient to handle the wide range of practical problems, numerous efforts have been made to combine different techniques together, for example, combining GA with local search[88, 89, 90], simulated annealing with local search[78, 72], controlled random search with local search[81]. However, there has been no general platform which can be used to facilitate the construction of hybrid algorithms. The USF is our attempt to provide such a platform. Basically, an optimization process is sampling process in which samples are continually taken from the parameter space until the optimization objective is met. An optimization algorithm is just a combination of sampling techniques which decide how the samples are taken. The USF is essentially a meta algorithm in the sense that it is not an optimization algorithm itself,

instead, it provides various sampling techniques as building blocks, such as, such as random sampling, pattern search and hillclimbing. For a specific problem, an appropriate optimization algorithm can be easily built by just combining some of these techniques according to the properties of the underlying problem. With the capability of correlating problems with appropriate search techniques, the USF is aimed to provide a general solution for various classes of optimization problems.

Another design objective of USF is to maximize the utilization of available computing resources. The existence of optimization algorithm is due to the limit of the available computing resources. If we have an infinite supply of computing resources which can evaluate any number of samples at one time, there will be no need for any optimization algorithm. A brutal enumeration method will find the exact global optimum. In practice, optimization is alway performed with a certain supply of computing resources, which may comprise a number of work stations, parallel computers, etc.. Therefore, the objective of an optimization algorithm is to optimally utilize the limited resources to obtain the desired solution quickly. In sequential optimization algorithms, the computing resource is considered a single monopolized resources, consequently they cannot fully utilize the resources with the parallel computing capability. Many parallel algorithms have been proposed to exploit the parallel computing resources. However, most of them are only designed for a certain hardware architecture, such as a certain parallel computer, and hence is not scalable to heterogeneous computing environment. The USF provides a resource management and allocation scheme which attempts to manage various computing resources in a unified framework and strives to take full advantage of available computing resource to maximize the optimization efficiency.

6.1 The Unified Search Framework

As discussed before, the USF includes various simple search techniques, such as random sampling and pattern search, as basic building blocks. For a practical problem, some building blocks are selected and run *in parallel*. Being coupled together with *memories*, which store certain samples during the optimization process, these building block can coordinate with each other to perform an efficient opti-

mization. The basic structure of Unified Search Framework is shown in Figure 6.1. There are two basic components in USF: *sampler* and *memory*. Any optimization

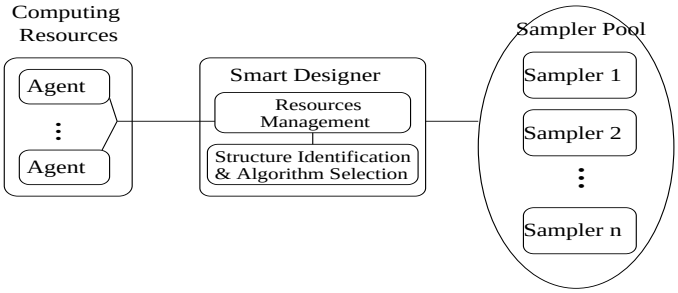


Figure 6.1: The Unified Search Framework

algorithm is established upon these two type of components. A sampler implements a certain search technique, such as, random sampling and pattern search, and a memory is used to store sampling points which are generated by a sampler and may be used in the future optimization process. For one specific problem, we can choose a few types of samplers and combine them together with the help of various types of memory for the most efficient optimization. Another important component of USF is its computing resource management. In one USF type optimization algorithm, the samplers are run in parallel with assigned allocations of computing resources. The computing resource management will maintain the resource usage of each samplers to its allocation. In the following, we will present the detailed description of these important components in USF.

6.1.1 Sampler

Samplers in USF implement various search technique and are building blocks of USF. Essentially a sampler performs a sampling process from the parameter space based on certain memories. For a specific problem, appropriate samplers should be chosen and combined based on the features of the problem. It is crucial for the efficiency of the optimization to include an proper selection of samplers which best exploit the structures of the underlying objective function. In fact, the selection of appropriate techniques is usually the most challenging and difficult task for a practical optimization problem.

For black-box problems, it is usually recommended to use direct search techniques[40], such as, Nelder-Mead simplex method[43] and pattern search. Each search technique tries to exploit a certain structure in the objective function and can perform well in the objective with the exploited structure. To select appropriate techniques, exploitable structures should be first investigated. In the following, we attempt to identify some important structural properties and the corresponding search techniques exploiting these properties.

Local correlativity is present in many practical problems. This structure means that in a parameter space, the neighboring points have similar function values, i.e., if a point has a good function value, the points near it may very possibly have good values and vice versa. Local search methods, such as, pattern search and downhill simplex, exploit this structure to find better points quickly. These methods normally start with a random point and look for a better point in the neighborhood of this point. If a new better point is found, the process is repeated at the new point until reaching the local optimum whose neighborhood does not include a better point. Because of its ubiquitous presence, local search techniques have been widely adopted in optimization algorithms.

Parameter separability is another structure which exhibits in many practical problems, especially large-scale problems. Parameter separability means that the objective function follows the following type:

$$f(\mathbf{x}) = \sum_{i=1}^{m} f_i(\mathbf{x}_i) \tag{6.1}$$

where $\mathbf{x}$ is a n-dimensional vector and $\mathbf{x}_i, i = 1 \ldots m$, are m lower-dimensional vectors whose elements are mutual exclusive subsets of elements of $\mathbf{x}$. With parameter separability, a large-scale problem can be decomposed into smaller sub-problems whose parameter spaces are subsets of the parameters of the original problem. We can optimize each $\mathbf{x}_i$ separately and then combine the results to find the global solution. The cross-over operation in genetic algorithms is one search technique which exploits this structure.

Mathematical models may also exist for practical problems. The methods exploiting this property fit a simple mathematical model to previous samples, and find the global optimum of this model and use it as a estimation of the global optimum in the original objective function. The model used in this type of methods is very important. On one hand, it has to be simple enough to make its own optimum-seeking easy to accomplish. On the other hand, it should also include enough complexity to approximate the underlying objective function as closely as possible. The model can be either deterministic, such as quadratic model, or stochastic, such as Weiner process. The appropriate selection of the model usually requires extensive *a priori* knowledge, which limits the application of this type of techniques.

Based on the structural properties and their suitable search techniques described above, we can make more sensible selection of search techniques. For the problem where sufficient *a priori* knowledge is available, we can take advantage of the knowledge to select proper search techniques. For example, if *a priori* knowledge strongly suggests that the response surface of the objective function is smooth and there is only one local extreme, a local search technique should be chosen and allocated with as much computing resources as it needs. If it has been known that most of the parameters are separable, the crossover technique should be selected and allocated with as much resources as possible.

For the problem where little *a priori* knowledge is known, a simple approach is just trial and error. That is, we can try some search techniques and choose those which perform better to continue. For example, in [91], GA and a local search method are combined and the local search is executed only if GA is either not increasing fast enough or if the GA is converging to a solution. A more advanced strategy is to first explore the parameter space and identify exploitable structures, and then try appropriate search techniques. For example, Torn[10] has proposed a conceptive strategy for multi-start local search algorithms.

- Take N random samples from the parameter space
- Start a local search for each sample as the starting point

- Analyze the characteristics of these local searches, such as, relative sizes of regions of attraction, minimizers;
- Based on this analysis, choose appropriate techniques and search parameters.

6.1.2 Memory

Memory selectively stores the sampling points output by samplers and these samples can be used by other samplers in later optimization process. The concept of memory is first proposed in tabu search algorithm[65], where memory is used to prevent the revisit of the sampling points already examined. Basically, any optimization algorithm can take advantage of memory to improve the efficiency. In USF, memory is of more importance than the one in tabu search and is the essential component of any optimization algorithm. In USF, samplers are run independently in parallel. To coordinately with each other, they have to be coupled with various types of memories. One example algorithm is shown in Figure 6.2. As shown in the figure, the algorithm is composed of two samplers: random sampling and pattern search. The samples generated by random sampling are put into a "drop-head" memory, which drops the oldest samples when the memory reaches its capacity. Pattern search uses these samples as the starting points to perform local search. Therefore, it basically implements a multi-start pattern search algorithm. In this algorithm, random sampling and pattern search are executed independently in parallel and their cooperation is achieved by the help of drop-head memory. Besides the drop-head memory used in the example, many other types of memory can be used. For example, we can use a memory which stores only the best few samples or the samples which satisfy a certain criterion, say, better than a certain threshold. The choice of of memories may affect the optimization efficiency substantially and should be carefully made based on the features of the underlying problem.

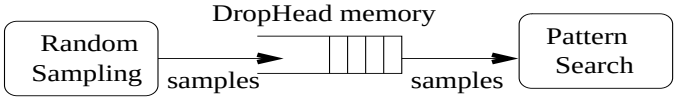


Figure 6.2: An example of samplers coupled with memory

With samplers and memories as building blocks, most of traditional optimization algorithm can be composed in the USF. For example, a genetic algorithm can be formed by combining random sampling, cross-over and mutation operators with a certain memory implementing the selection mechanism. Furthermore, it is easy to build an optimization algorithm which is specifically tailed for a certain problem. The algorithms implemented in USF have an additional advantage, i.e., the samplers are performed in parallel to take full advantage of available computing resources and the coordination between samplers can be easily controlled by adjusting memory types and resource allocations. We will show in the tests that these flexibility can improve the optimization efficiency significantly.

6.1.3 Computing Resource Model and Management Mechanism

As mentioned before, the computing resource management of USF is based upon the assumption that function evaluation consumes most of computing resources while the consumption of the other operations can hence be ignored. Furthermore, we also assume each function evaluation consume the same amount of computing resources. Therefore, the resource allocation in USF is performed based on the number of function evaluations. In other words, we calculate the resource usage of each sampler by the number of function evaluation which has been executed. Note that these assumptions are valid for most of practical optimization problems where function evaluations are usually performed with complicated computer simulations. In fact, the number of function evaluations have been widely used in optimization literature as the approximate measure for the computing effort of one optimization algorithm consumes in benchmark tests. In the following, we describe the details of this resource management mechanism.

The resources management of USF is illustrated in Figure 6.3. The computing resources can be composed of multi-processor parallel computers or a network of workstations. On each computing device, an agent is run to manage the device and communicate with the resource manager of USF. Whenever an agent finds that the managed computing device is free, it requests an experiment, i.e., function evaluation, from the manager. Each sampler maintains an experiment queue. Whenever

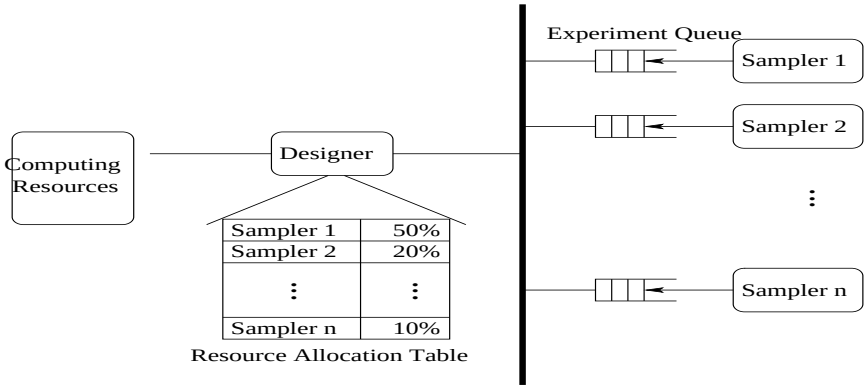


Figure 6.3: Resource allocation mechanism

the sampler generates a sample, it puts it into the queue for function evaluation. When the queue is full, the sampler will stop generating new samples until the queue becomes available again. When receiving a request from an agent, the resource manager will then examine the experiment queues of all running samplers and choose one to return according to the resource allocation rule.

To allocate computing resources among the samplers, the manager maintains a resource allocation table which defines the percentage of computing resources that each sampler can use. For example, the following table shows the resource allocation among 3 samplers:

Sampler 1	50%
Sampler 2	20%
Sampler 3	30%

This table indicates that Sampler 1 may use 50% of resources, Sampler 2 20% and Sampler 3 30%. The resource allocation table should be decided based on the features of the underlying problem and can be adaptively adjusted during the optimization process to maximize the efficiency.

The manager also maintains another table which records the actual resource usage for each sampler and is initialized to be the same as the allocation table in the beginning. Whenever the manager receives a experiment request from computing resources, it will check those samplers with non-empty experiment queues, and choose the one with the maximum unused allocation in the usage table. Af-

ter sending the selected experiment to computing resources, the manager updates the usage table by subtracting the chosen item by 1 and then adding each item in the usage table (including those with empty queues) by its corresponding allocation percentage. One example of this procedure is shown as in Figure 6.4. As shown in

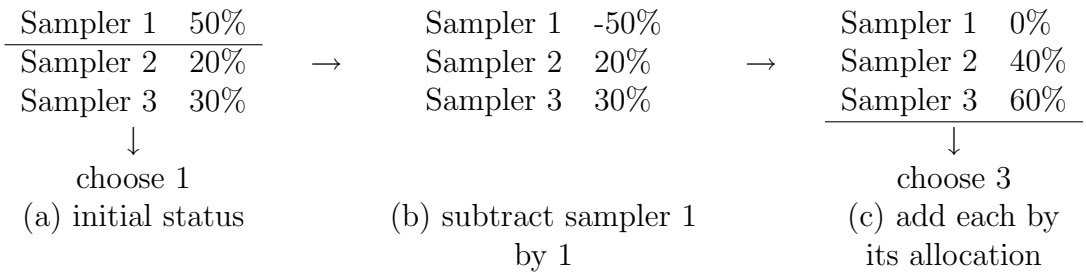


Figure 6.4: An example of resource allocation operation

the example, the usage table is first initialized as the allocation table. Suppose the experiment queues of all samplers are not empty, sampler 1 is first chosen to send its experiment for evaluation since it has the maximum allocation. Then the usage of sampler 1 is subtracted by 1 as shown in table (b) and the usage of each sampler is added by its allocation as shown in table (c). After this, sampler 3 will be chosen next time since now it has the maximum value in the usage table.

By using the above method, the computing resources can be distributed among the samplers in accordance with the resources allocation table. The resource allocation table provides us with the flexibility to adjust the coordination between samplers running in parallel and achieve the best efficiency. In fact, many sequential optimization algorithms use certain mechanisms to adjust the computing resource allocation between search techniques. For example, in simulated annealing[13], the "temperature" parameter is used to control the balance between random walk and hillclimbing. With the high temperature in the beginning, random walk runs more frequently and hence uses more resources. With the temperature cooling down, hillclimbing gets more and more computing resources. In genetic algorithm, similar control parameters also exists, such as, crossover rate or mutation rate.

Another advantage of the above resource allocation mechanism is that if some sampler can not use all of its allocation, the surplus resources will be consumed by other samplers whose allocation do not meet their needs. Some samplers may

not generate enough samples to use up its allocation. For example, a local search sampler has to wait for the results of previous samples to decide further samples. Therefore, its experiment queue may become empty during the waiting period. Since the allocation mechanism in USF only examines the items with non-empty queues, its allocation will be used by other samplers. By such "borrowing mechanism", the full utilization of the available computing resource can be achieved. For example, a random sampling sampler with a very small allocation, say 0.0001%, can be always used to take up all the surplus resources. Since its allocation is very small, it will hardly affect other samplers' operation and only come to execution when other samplers could not fully utilize the available resources.

6.1.4 Parallel Optimization Strategy

One design objective of USF is to fully utilize available computing resources. To achieve this, All samplers in USF are run in parallel and their function evaluations are distributed to computing resources based on their allocation. As mentioned before, some search technique, especially local search methods, such as, hillclimbing, may not be able to fully use its allocated resources. For these techniques, a certain parallel optimization strategy has to be used for full resource utilization. Many parallel optimization algorithms has been proposed[92, 93]. The parallel strategies used in these algorithm can be classified into the following three categories:

Problem partition divides the parameter space of the original problem into many small areas and performs optimization on each of these sub-spaces in parallel. The disadvantage of this strategy is that since each area is allocated with equal shares of computing resources, it may waste resources on many trivial areas which is very unlikely to have the global optimum.

Multi-path is a method used to parallelize multi-start local search methods. Basically, this strategy just performs multiple local searches with different starting points in parallel.

Algorithm parallelization is to parallelize a sequential algorithm by careful changing its design. This method is specific to a certain search technique and hence

is not generally applicable to other techniques. Furthermore, its parallelity is normally constrained by the inherent limit of the algorithm and not scalable with available computing resources.

In USF, two mechanisms have been used to achieve the complete resource utilization. One is the "borrowing mechanism" described in the previous section, i.e., when one sampler cannot use up its allocation, the surplus can be temporarily used by other samplers. For example, one simple method to take advantage of all computing resources is to include random sampling with a very low resource allocation, which will use up the resources left by other samplers. In addition, USF use multi-path method described as above to automatically increase parallelity of search techniques. In USF, resource manager keeps track of the demand and supply of computing resources. When resource manager finds that the demand of a sampler is always less than its allocation, i.e., its experiment queue is always empty, it will increase the number of this type of samplers and run these samplers in parallel to use up its allocation. Note that in USF, the number of "multi-path" is decided by available computing resources instead of a predefined level in traditional multi-path methods.

6.2 Test Results

In this section, we use benchmark tests to demonstrate the flexibility of USF in handling various situations. Each test is repeated for 50 times and the average is taken as the final result.

6.2.1 Effect of Memory Types

One of the main advantage of USF is that it provides various building blocks and can be easily used to built an optimization algorithm by combining appropriate building blocks. Furthermore, the coordination of these samplers can be achieved with various types of memories. The tests in this section will demonstrate that using appropriate types of memory can greatly improve the optimization performance. Memory is used in USF to store previous samples and couple samplers together.

Various memories can be used for different problems. The tests in this section have examined two types of memory used:

- Drop-head, which drops the oldest samples when its capacity is reached.
- Drop-worst, which drops the worst samples when its capacity is reached.

Two samplers are used in the tests: random sampling and pattern search and they are coupled together with a certain memory just like the example shown in Figure 6.2. The optimization algorithms obtained with the above memories are tested on a benchmark function, i.e., 20-dimensional Rastrigin function. The convergence curves of two memory types are shown in Figure 6.5. We can see that by using drop-worst memory, the optimization efficiency can be improved substantially.

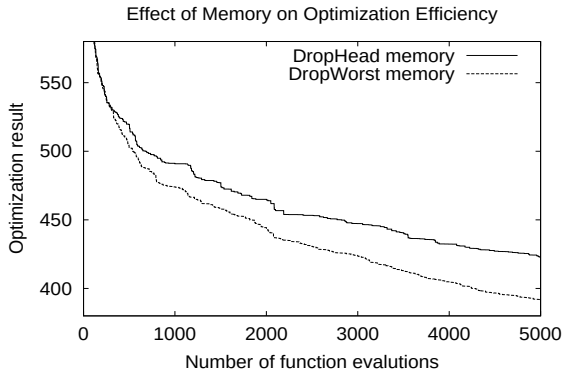


Figure 6.5: Effect of memory on optimization efficiency

6.2.2 Effect of Resource Allocation Strategy

In addition to memory, USF also provide a flexible resource allocation mechanism to adjust the coordination of samplers. In this section, we will examine the effect of resource allocation strategy on the optimization efficiency. We still use the same two samplers as before: random sampling and pattern search, however, we vary the resource allocation between these two samplers in the tests. The results are shown in Figure 6.6. The horizontal axis indicate the resource allocation of random sampling. Given the allocation of random sampling, pattern search will take all the remaining resources with multi-path strategy described before. For example,

if random sampling gets 20% resources, pattern search then gets 80%. The vertical axis indicates the optimization result after 5000 function evaluations. As we can see from the figure, the optimization efficiency varies greatly with the resource allocations for the samplers. At one extreme, allocating all computing resources to random sampling is equivalent to a pure random sampling algorithm. At the other extreme, allocation most of resources to pattern search is equivalent to a simple multi-start pattern search algorithm. Neither case can produce good efficiency in the test. The best balance between two samplers is achieved at a point between two extreme case. As shown in the figure, allocating 60% resources to random sampling produces the best efficiency.

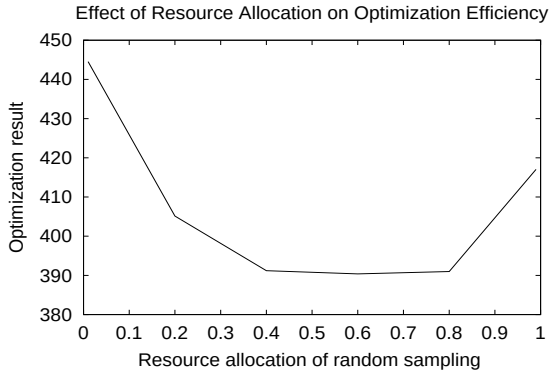


Figure 6.6: Effect of resource allocation on optimization efficiency

6.2.3 Scalability of Parallel Optimization

One important design objective of USF is to fully utilize available computing resources. In this section, we examine if the optimization efficiency can be improved by making full use of computing resources. In the tests, we use the algorithm described in Figure 6.2 with a network of workstations to optimize 20-dimensional Rastrigin function. We vary the number of workstations in the tests and compare the optimization performance of USF with different number of workstations. To simulate the situation where function evaluations are expensive, we did not code the benchmark function directly into the optimization algorithm. Instead we used a slow script language to evaluate the benchmark function. The left plot in Figure 6.7 shows

the optimization results as a function of elapsed time for 1, 2, 4 and 8 workstations. We can see with increasing computing resources, the optimization performance is improved accordingly. If we specify a function value as the optimization objective, for example, 480 in the left plot, we can see that the optimization time required to achieve this objective is approximately reduced proportionally with the number of workstations as shown in the right plot. That is, the linear speed-up may be achieved with the USF parallel optimization mechanism. Note that the speed-up is dependent on the underlying problem and the selected search techniques. Although the linear speed-up may not be alway be obtained, the advantage of resource management mechanism in USF is its scalability, i.e., it can always take full advantage of available computing resource to improve the optimization efficiency.

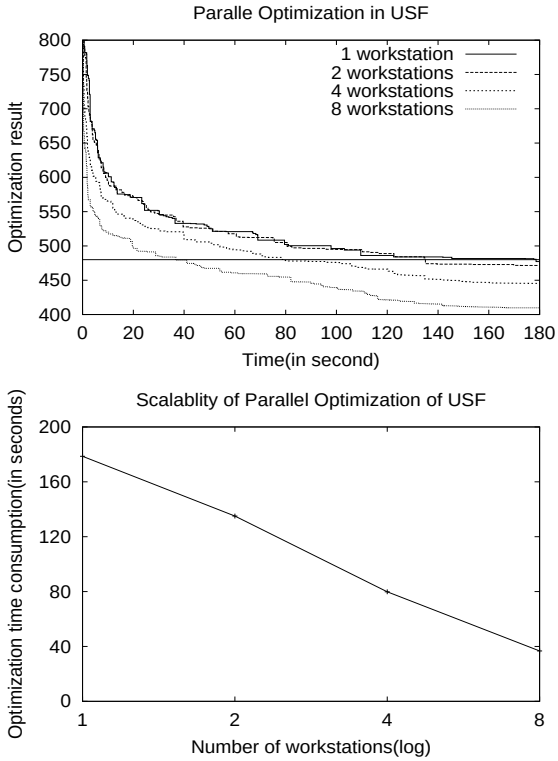


Figure 6.7: Scalability of parallel optimization in USF

6.3 Conclusion

The Unified Search Framework provides a flexible platform to construct optimization algorithms for various practical optimization problems. To achieve the best efficiency, appropriate search techniques should be first chosen. In addition, the coordination among these techniques should also be carefully adjusted. This can be achieved in USF by adjusting resource allocation of these techniques and selecting memory types coupling these techniques. To make the correct selection and adjustment, the features of the problem have to be carefully examined. Currently, this step has to be performed manually based on the practitioner's experience or trial and error. The USF made some attempts to establish the correspondence between search techniques and their suitable structural properties. However, more work has to be done to automate this process. Essentially, in addition to establishing the correspondence between search techniques and suitable structures, we have to know how to identify these structures and how they affect the coordination of search techniques.

CHAPTER 7
Application to Network Optimization

With the capabilities provided by the on-line simulation scheme, any network protocol can be optimized to varying network scenarios. However, Formulating the tuning of the concerned network protocol as an optimization problem often is not a straightforward task. Since network performance is measured by many metrics, such as network utilization, queueing delay and packet loss, it is very important to design a proper optimization metric which best captures the relationship between the network protocol and performance metrics. A poor problem formulation may seriously affect the performance of the on-line tuning scheme. In the following, we investigate the on-line tuning problems of three important protocols: RED queueing management, OSPF routing protocol and BGP routing protocol.

7.1 Optimize RED for Network Congestion Control

Congestion control in the current Internet is accomplished by end-to-end congestion avoidance together with queue management mechanism. Traditional DropTail queue management could not effectively prevent the occurrence of serious congestion and often suffer from long queueing delays. Furthermore, the global synchronization may occur during the period of congestion, i.e., a large number of TCP connections experience packet drops and hence back off their sending rate at the same time, resulting in underutilization and large oscillation of queueing delay. Random Early Detection (RED) has been proposed [94] to address these problems. The basic idea of RED is to detect the inception of congestion and notify traffic sources early to avoid serious congestion. It has been demonstrated to be able to avoid global synchronization problem, maintain low average queueing delay and provide better utilization than DropTail[94]. Therefore, IETF has recommended RED as the single active queue management for wide deployment in the Internet[95]. However, the setting of RED parameters has proved to be highly sensitive to network scenarios and the performance of misconfigured RED may suffer significantly [96, 97, 98]. In

addition, since network is a dynamic system, RED needs constant tuning to adapt to current network conditions. In view of this, it has been debated whether or not RED can achieve its claimed advantages[98, 99, 100].

Currently the interaction between RED and TCP is not yet clearly understood. Based on simplified models, some general guidelines for setting RED parameters have been proposed[94, 97, 101]. Intuitive modifications on RED have also been proposed to automate the tuning of RED under varying network conditions by adjusting one of the parameters[96, 102]. However, the effectiveness of these methods in complex network scenarios is still under investigation. Rather than relying on simplified models or intuition, we will attempt to use the on-line simulation scheme to perform the dynamical tuning of RED.

7.1.1 Formulation of RED Optimization Problem

7.1.1.1 Parameter Sensitivity of RED

RED uses the average queue size $\bar{q}$ as an indicator of the congestion extent and determines the packet drop rate accordingly. Fig 7.1 illustrates the working mechanism of RED. As shown in the figure, the instantaneous queue size q is sampled

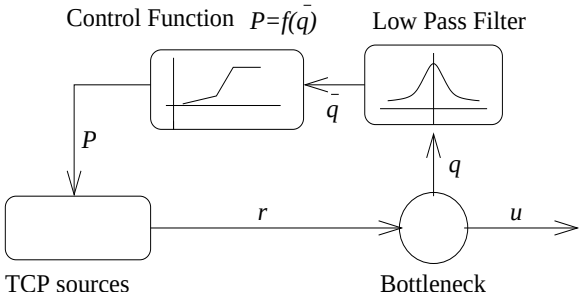


Figure 7.1: RED working mechanism

at every packet arrival and then passed through a low-pass filter to remove transient noises. Based on the smoothed average queue size $\bar{q}$, the drop probability P is calculated with a control function $P = f(\bar{q})$. The arrived packets are randomly dropped (or marked) according to this probability P. Traffic sources react to these drops and adjust offered load r accordingly. Therefore, RED is mainly designed to work with TCP traffic sources which are responsive to packet drops and it will not

work well in the cases like UDP traffic or short-life HTTP traffic.

A queue will build up and keep increasing if the offered load is larger than the bottleneck capacity; therefore, the objective of a queue management is to stabilize the offered load around the bottleneck capacity. Basically, TCP sources increase their sending rate every round trip time; on the other hand, the packet drops cause TCP sources to lower their sending rates. In the equilibrium status, the increase rate of TCP traffic should be approximately equal to its decrease rate caused by packet drops and thus the offered load will stabilize around a certain level. If this equilibrium status is achieved while maintaining a certain queue size, the link utilization will be close to 1, i.e., the offered load will stabilize around the bottleneck capacity. The rationale of RED is to search for an appropriate packet drop rate by varying the average queue size to counteract the increase of offered load.

There are four parameters in RED. Among them, the moving average weight w_q determines the cut-off frequency of the low-pass filter, and the other three parameters, i.e., minimum threshold min_{th}, maximum threshold max_{th} and maximum drop probability max_p, determine the control function $P = f(\bar{q})$. In the standard version of RED, the control function is determined by the parameters as illustrated in Fig 7.2. With this function, the drop probability can be calculated according to

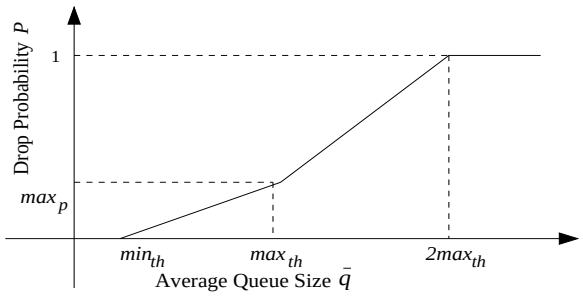


Figure 7.2: RED control function $P = f(\bar{q})$

the average queue size. The equilibrium drop probability depends on two factors, the offered load increase rate and the granularity of congestion notification, i.e., the load decrement caused by one packet drop. With TCP fast recovery and fast retransmission mechanism, each drop will cause a TCP source to decrease its sending rate by half. Therefore, the granularity of the congestion notification is determined

by the average TCP sending rate. When the average sending rate is large, for example, a small number of TCPs share a bottleneck, each packet drop will cause a large decrease in offered load, and *vice versa*. In different scenarios, the increase rate of offered load is also different. For example, the increase rate will be large when there are many TCP flows or the round trip time is short. As a result, the drop probability should be adjusted according to network scenarios to maintain a stable equilibrium point. If the control function remains unchanged, the average queue size has to be varied to obtain the new equilibrium drop probability. Therefore, to keep the average queue size stable around a certain level in varying conditions, the control function has to be adjusted accordingly, i.e., the three parameter which determines $f(\bar{q})$ should be dynamically tuned.

w_q controls the cut-off frequency of the low-pass filter. The cut-off frequency should be high enough to detect manageable traffic variations, while low enough to filter out transient traffic oscillations which can not be effectively controlled by RED. For example, the oscillation within one round trip time rtt should be removed. Therefore, the optimal w_q is usually related to rtt. In addition, since the average queue size is calculated at every packet arrival instead of a constant interval, different link speeds will result in different packet arrival intervals and hence affect the cut-off frequency of the low-pass filter. Consequently, the optimal w_q is also dependent on the link speed.

7.1.1.2 Optimization Objective

For a queue management mechanism, there are basically two performance metrics, i.e., link utilization and average queue size. The main objective of RED is to *maintain a high utilization while keeping a low average queue size*[94]. However, optimizing one of the performance metrics may compromise the other. For example, a high link utilization can always be obtained by increasing min_{th} or decreasing max_p, hence virtually increasing the average queue size. On the other hand, a low average queue size can be obtained by decreasing max_{th} or increasing max_p. However, this obviously will cause underutilization of the link. Therefore, an appropriate tradeoff has to be made to reflect the requirement of network operators. This is essentially a

multi-objective optimization problem and corresponding techniques should be employed to convert it into a tractable single objective problem.

One classic multi-objective optimization technique is to optimize the weighted average of the performance metrics. The weights for different metrics reflect the quantitative tradeoff among them and are essential to the effectiveness of optimization results. However, the weights are normally difficult to determine. Another common technique is to define the lower limits for less significant metrics, and only optimize the most important one with the restriction that the other metrics are not below their limits. Instead of using traditional multi-objective optimization techniques to directly work on link utilization and queueing delay, we have proposed a performance metric whose optimization will cause RED to settle in a equilibrium status and hence achieve high utilization and low queueing delay.

As mentioned above, in the equilibrium status, the average queue size of RED stabilizes around a certain level. When traffic pattern changes, the equilibrium point may also shift which makes the average queue size move around. This is an undesired behavior since end users normally expect a predictable delay and constant changes in delay are unacceptable for delay-sensitive applications. Furthermore, when the average queue size drifts beyond the control of RED, RED will become unstable, i.e., the queue status oscillates between full and empty[96, 97]. This not only causes end users to experience significant delay jitters, but also results in link underutilization. Therefore, it is important to keep the average queue size of RED stable at a target level, such as the middle between min_{th} and max_{th} as proposed in[102]. In consideration of this, we define the performance metric to be optimized as:

$$m = \frac{\sum_{i=1}^{N}(\bar{q}_i - q_0)^2}{N} \tag{7.1}$$

where q_0 is the expected average queue size predefined by network operators, $\bar{q}_i$ is the periodic sample of the average queue size and N is the number of samples. This metric essentially calculates the variance of the average queue size relative to q_0 over a certain period of time. When the equilibrium level of RED is far from the expected level, m will be large. Or when RED is misconfigured and hence the equilibrium cannot be reached, the queue size will oscillate greatly, also resulting in

a large m. Therefore, minimizing m will cause RED to avoid both situations and always maintain an equilibrium around q_0. Thus, high link utilization and stable queueing delay can both be achieved.

Based on the above analysis, the optimization of RED can be formulated as: given a parameter space D specifying the ranges of parameters, minimize the following objective function

$$m = f(w_q, min_{th}, max_{th}, max_p) \tag{7.2}$$

where, m is the performance metric to be optimized, i.e., the variance of RED queue size relative to the expected level, $f(\cdot)$ is a scalar function mapping a set of RED parameters to the performance metric. This function is determined by the specific network scenario and is analytically unknown, which is the basic feature of black-box optimization. For a certain RED parameter setting, the value of m can be empirically evaluated with network simulation based on Equation (7.1).

7.1.2 Simulation Results

7.1.2.1 Simulation for Optimization of Single RED

This section presents simulations of the proposed on-line tuning approach, which deal with varying traffic load and round trip time, two major factors affecting RED performance.

The network topology used in the simulations is shown in Fig 7.3. We used

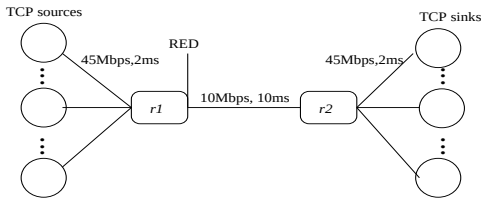


Figure 7.3: Network topology for simulation

ns[3] as the simulation tool. Infinite FTP traffic between TCP sources and sinks is generated to build up a queue at router $r1$. RED is configured on $r1$ to manage a 100-packet buffer. Each simulation runs for 40 seconds and network conditions are

changed twice during the simulation. We will compare the performance of standard RED and RED controlled with the on-line simulation framework under changing network conditions.

We define an expected average queue size of 30 packets and the objective is to maintain the equilibrium status of RED around this level. According to the common guideline of RED parameter setting, we use $min_{th} = 15, max_{th} = 45, max_p = 0.1, w_q = 0.002$ for standard RED. We also assume that the on-line simulation system can promptly detect the change in network conditions and trigger the optimization process of RED parameters. In reality, this can be achieved by monitoring the change in performance metrics or analyzing traffic statistics directly.

First we test the tuning of RED to varying traffic load. The number of TCP flows in the simulation starts with 16, then increases to 64 after around 13 seconds, and finally decreases to 4 after another 13 seconds. The instantaneous queue sizes of standard RED and RED with on-line simulation control are shown in Fig 7.4. The

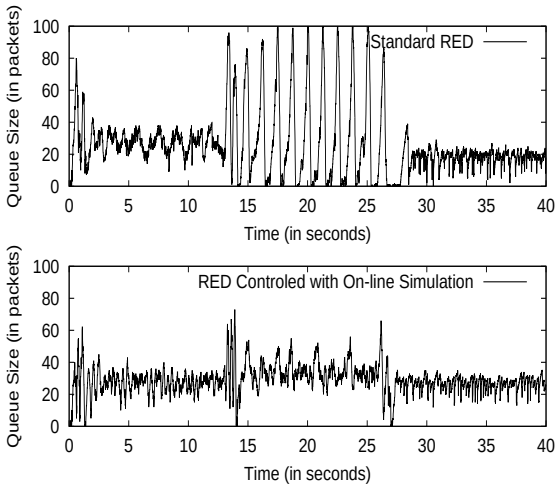


Figure 7.4: Comparison of standard RED (upper graph) and RED controlled by on-line simulation (lower graph) under varying traffic load

upper graph shows that for the standard RED, when the traffic load increases beyond the control of current RED parameter setting, the equilibrium status may not be broken and the queue remains in a very unstable status where large oscillations between full and empty queue persist. On the other hand, when the traffic load decreases to a certain level, the queue frequently becomes empty and this causes the

underutilization of the link capacity. The lower graph shows that when dynamically tuned, RED always maintains an equilibrium status where the queue size remains very stable and the utilization is close to 100%.

Then we test the tuning of RED to varying round trip time. The simulation starts with 16 TCP flows and each with a round trip time of 18ms (not including queueing delay). After 13 seconds, the *rtt* of these flows is increased to 170ms. And after another 13 seconds, the *rtt* is reduced to around 2ms. The instantaneous queue sizes of standard RED and RED with on-line simulation control are shown in Fig 7.5. The upper graph shows that when *rtt* is increased to 170ms, the equilibrium

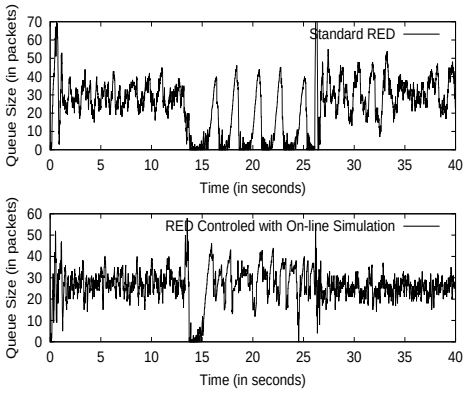


Figure 7.5: Comparison of standard RED (upper graph) and RED controlled by on-line simulation (lower graph) under varying round trip time

of standard RED queue is again broken and the queue keep oscillating between full and empty status. And when *rtt* is reduced to 2ms, although the queue does reach an equilibrium status, there still exist big variations in queue size. As shown in the lower graph, the dynamically tuned RED eliminated these problems.

7.1.2.2 Real Network Experiment for Optimization of Multiple RED Queues

The effectiveness of our approach is also tested with experiments in real network situations. This section presents one such experiment. A Linux-based testbed shown in Fig 7.6 is used and *ns* is adopted for network simulation in the on-line simulation system. There are 4 Linux routers in the network and each of them

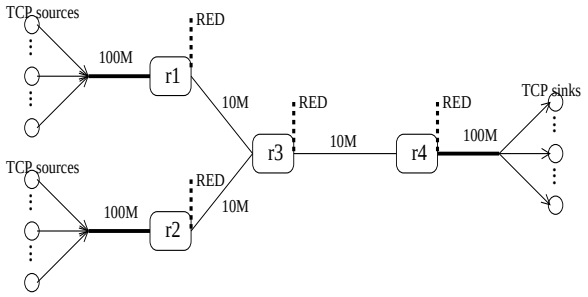


Figure 7.6: Linux-based testbed topology with multiple RED queues

is configured with a RED queue which is monitored and controlled by the on-line simulation system through SNMP. Again, infinite FTP sources are used to generate network traffic. Note that in this test we will try to tune the parameters for all four RED concurrently. Since optimizing each RED individually may compromise the performance of the others, we have taken all RED queues as a single black-box system with a total of 16 parameters. Consequently, a global performance metric has to be defined based on the objective of network operators. If using ISP-based metrics, such as utilization and queueing delay, a certain multi-objective technique has to be employed to combine the metrics from every RED router. Instead, we have selected an end user performance metric, i.e., the coefficient of variation ($\frac{\sigma}{\mu}$) of goodputs for TCP connections, which measures the variability of TCP goodputs. This choice is somewhat arbitrary, only to demonstrate the effectiveness of our approach. In addition, choosing such a metric is also to demonstrate the flexibility of the approach, i.e., rather than being restricted to a few metrics like utilization and delay, RED can be tuned according to any performance metric defined by network operators though the mechanism of how RED affects this performance metric may be completely unknown.

During the experiment, a number of TCP flows are generated from one side to the other. The goodputs of these TCP flows are collected periodically from TCP sinks. The Coefficient of Variation(COV) of the goodputs is calculated and plotted as a function of time as shown in Fig 7.7. In the beginning, the parameters of these RED queues are set to *random* values to represent a misconfigured system, which results in a large unfairness between TCP flows, i.e., a high average COV value and large oscillations. At 325 second, the on-line simulator starts and detects the

misconfiguration of REDs. Soon the good configuration with a performance better than a predefined threshold is found and the network is reconfigured. This results in an immediate performance improvement as shown in the plot: the average of COV drops to a very low value and the instantaneous COV curve becomes stable over time.

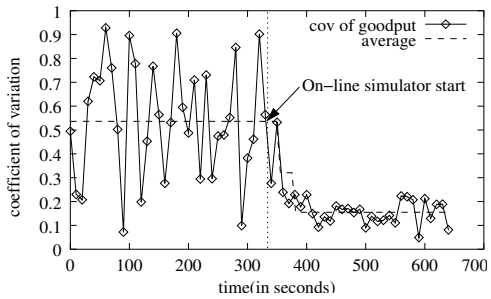


Figure 7.7: Tuning multiple RED queues for optimizing coefficient of variation of goodputs

7.1.3 Conclusion

Simulation results have demonstrated that when managed by the proposed approach, RED can effectively control congestion under varying network conditions and achieve its design objective. In addition to the on-line tuning, optimization techniques can also be used for the study of network protocols. The obtained empirical knowledge can help with better understanding of network protocols. For example, we can optimize RED for different network scenarios and obtain the correlations between RED parameters and network conditions, such as the correlation between w_q and round trip time.

7.2 Optimize OSPF for Traffic Engineering

In this section, we address the problem of traffic engineering in a network of OSPF routers. Traffic engineering is defined as the task of mapping traffic flows onto an existing physical topology to meet the objectives of network operators. In current Internet, IP traffic is mapped onto the network by standard routing protocols, such as, Open Shortest Path First (OSPF) for intra-domain traffic and Border Gateway Protocol (BGP) for inter-domain traffic. When routing the traffic,

the routing algorithms used in these protocols normally select the shortest path without taking into account the traffic conditions and Quality of Service (QoS) constraints. These routing protocols are often called *topology-driven*. The routing generated by such algorithms tends to result in a highly uneven mapping of traffic. Some links may get very congested while the others may be consistently under-utilized. This phenomenon has been confirmed by many traffic measurements[103, 104] where a large variation in link utilization is observed across the network. Traffic Engineering (TE) tries to eliminate this situation by adapting the network routing according to the prevailing traffic conditions.

Two main approaches have been taken to solve the traffic engineering problem in the Internet. One approach is to deploy the emerging MPLS technology which is not constrained by the shortest path nature of routing. Constraint-based routing can be used to compute routes in an MPLS network subject to QoS and policy constraints. Another approach is to adjust the link weights of the existing network (running OSPF) such that the OSPF routing with these link weights leads to desired routes [28].

The main issue with using existing OSPF routing for traffic engineering is its shortest path nature. OSPF routes traffic on shortest paths based on the advertised link weights. As a result, the link along the shortest path between the two nodes may become congested while the links on longer paths may remain idle. OSPF also allows for Equal Cost Multi Path(ECMP) where the traffic is distributed equally among various next hops of the equal cost paths between a source and a destination [105]. This is useful in distributing the load to several shortest paths. However, the splitting of load by ECMP is not optimal as shown in [106]. Various methods have been proposed in literature to balance the traffic across the network in OSPF routing framework. One of the earlier approaches was to adapt link weights to reflect the local traffic conditions on a link or to avoid congestion ([107, 108, 109]). This is called adaptive routing or traffic-sensitive routing. However, adapting link weights to local traffic conditions leads to frequent route changes and is unstable (see [110, 111] for stability analysis). Additionally, adaptive routing is based on the local information and therefore cannot optimize traffic allocation from the viewpoint of

the overall network. These drawbacks are alleviated in [28] where the traffic demand of the network is used to estimate the offered load for each link and then a local search heuristic is deployed to find "good" OSPF link weight settings which optimize the traffic load allocation across the network.

Basically, authors in [28] have modeled the optimum setting of OSPF weights as a global optimization problem. They have chosen a heuristic cost function which is piecewise linear with offered load. By using such a cost function, they can model the optimal general routing as a linear programming problem and solve for the exact solution. Here the optimal general routing represents routing where there is no limitation on the way a flow is split among multiple paths available between a source and destination. The optimal general routing is the best that can be achieved by carefully setting up multiple Label Switched Paths (LSPs) in MPLS. Authors in [28] have shown that for the proposed AT&T WorldNet backbone, OSPF with optimized link weights can yield a routing with a performance within few percent of the optimal general routing and even for the randomly generated network topologies, 50%-110% more demand can be supported than link weight setting based on some standard heuristic.

In this work, we will apply the on-line simulation scheme to the optimization of OSPF link weights. We have chosen the total packet drop rate in the network as the optimization metric since it is a more accurate to indicate the congestion in the network than the heuristic metric in [28] and it also has great impacts on the performance of some underlying protocols, such as TCP. The packet drop rate for one set of link weights can be estimated using packet-level or flow level simulation. However, in this work, we use an analytic approach to calculate the packet drop rate by using a GI/M/1/K queuing model. This is considerably faster than the simulation approach.

7.2.1 The Objective Function

Our goal is to minimize the packet drop rate in the network for a given mean and variance of the aggregate demands between each source and destination routers. Let us consider a network represented by a directed graph $\mathcal{G}$=($\mathcal{N}$,$\mathcal{L}$), where $\mathcal{N}$ and

$\mathcal{L}$ represent respectively the set of routers and links in the network. Each link $l \in \mathcal{L}$ has bandwidth denoted by B_l and a buffer space of K_l packets. We assume that packets arriving when the buffer space at a link is full are dropped and there is no other active queue management algorithm running at the routers. In addition to the knowledge of bandwidth and buffers at all the links, we assume that an estimate of the mean and variance of the aggregate demand from each source s to destination t is known. Let $\mathcal{D}$, $\mathcal{V}$ denote the mean and variance matrix of the estimated aggregate demand. In practice, all such information can be obtained using the tools described in [112, 113].

In the following, we will first show how to derive the drop probability for one link based on the offered load. Then we will formulate the optimal general routing problem which aims to optimize the overall packet drop rate for the network. Note that the OSPF optimization problem is just the optimal general routing subject to the shortest path constraint.

7.2.1.1 Link Drop Probability

Let P denote the packet drop probability on a link, λ, σ^2 denote the mean, variance of the offered load to this link in packets per second, and B, K denote its bandwidth and buffer space respectively. In order to find a closed-form expression for the packet drop probability P, let us assume an exponentially distributed packet size with mean $\bar{X}$. However, we consider a general arrival process. We compute the packet drop probability at the link using a GI/M/1/K queuing model. The drop probability of a finite GI/M/1/K has been approximated by an infinite buffer GI/M/1 queue [114] using the following equation.

$$P(N_K = K) = \frac{P(N_\infty = K)}{P(N_\infty \leq K)} \tag{7.3}$$

N_K denotes the number of packets in the finite buffered queue, whereas, N_∞ denotes number of packets in the infinite buffer GI/M/1 queue. The queue length distribution of GI/M/1 queue is given by [115]:

$$P(N_\infty = j) = A\omega^{j-1} \qquad (j \geq 0) \tag{7.4}$$

where A is the normalization constant and ω is a constant depending on the arrival process and service rate. ω can be obtained by solving the following equation:

$$\omega = \gamma\left((1-\omega)\mu\right) \tag{7.5}$$

where $\gamma(s)$ is the Laplace transform of the arrival process and μ is the service rate which is given by $\frac{B}{X}$. In order to solve (7.5) for ω, we need to assume a inter-arrival time distribution for the arrival process. Let us consider the Generalized Exponential (GE) distribution for modeling the arrival process to first two moments. We discuss below the reason for choice of GE distribution.

The pdf of GE distribution is given by

$$g(x) = (1-p)\delta(x) + pae^{-ax} \tag{7.6}$$

where $\delta(x)$ is the delta function, p and a two constant parameters. As can be seen from (7.6), a GE process is characterized by two parameters, p and a. GE distribution is a special case of H_2 distribution and can be used to model general inter-arrival processes that are more bursty than Poisson process. For a Poisson process the variance is equal to the square of mean. Hence, GE distribution may be used to model the first two moments of processes with variance greater than the square of mean. If the arrival process is represented by a GE distribution, then, with probability p the inter-arrival time is exponentially distributed with mean a and with probability $1-p$, the inter-arrival time is zero. Hence, this distribution represents a batch arrival process with geometrically distributed batch size and exponentially distributed inter-batch arrival times. For a link with λ, σ as its mean and variance of the offered load, we can have the parameters of the GE distribution representing the arrival process:

$$p = \frac{2\lambda^2}{\sigma^2+\lambda^2} \text{ and } a = p\lambda \tag{7.7}$$

The merging of N independent GE(p_i,a_i) processes is a bulk-arrival Poisson process with mean arrival rate a equal to $\sum_{i=1}^{N} a_i$ and p equal to $a/\sum \frac{a_i}{p_i}$. Similarly, splitting of a GE(p,a) process into N streams according to a Bernoulli filter $r_1, r_2, ...r_N$, the

parameters of the i^{th} process are

$$p_i = \frac{p}{p(1-r_i)+r_i} \text{ and } a_i = r_i a. \tag{7.8}$$

Reader may refer to [116], Section 1.4 for more details.

The packet arrival process of a single TCP flow is bursty in nature with a "bulk" of packets arriving every round-trip time. The model that we have considered implies that we have "bulk" arrivals (in form of bursts of packets from competing TCP sources) of varying sizes arriving into a queue. Our model does not capture the feedback effect of packet drops on TCP flows because we have considered the aggregate traffic arriving at an OSPF router as our demand estimate.

Taking the Laplace transform of (7.6), we get,

$$G(s) = 1 - p + \frac{pa}{s+a} \tag{7.9}$$

Then substitute it into (7.5) and solve it for ω for the GE arrival process gives

$$\omega = \rho + (1-p) \tag{7.10}$$

where,

$$\rho = \frac{a}{\mu} = \frac{a\bar{X}}{B}. \tag{7.11}$$

Finally, using (7.3), (7.4), (7.5) and (7.9), we get the packet drop probability

$$P = \frac{(p-\rho)(\rho+1-p)^K}{1-(\rho+1-p)^{K+1}} \tag{7.12}$$

In summary, (7.12) represents the closed form expression of packet drop probability, P, on a single link as a function of mean, variance λ, σ^2 of the arrival process, mean packet size $\bar{X}$, link bandwidth B and buffer space K. Figure 7.8 shows the drop probability as a function of the offered load for difference values of variance of the inter-arrival time for a buffer size of 20 packets. As expected, higher drop probability is observed when the arrival process has a high variance, i.e., when the incoming traffic is more bursty.

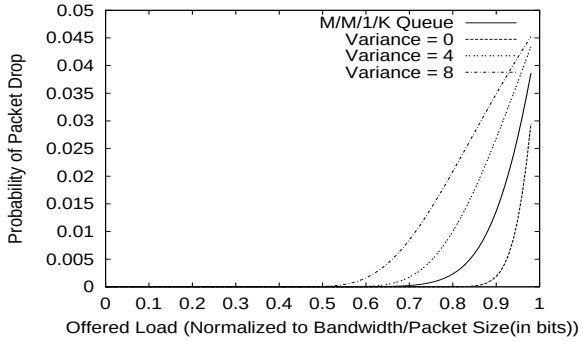


Figure 7.8: Packet drop probability as a function of offered load for a GE/M/1/20 queue for different values of variance

7.2.1.2 The Optimal General Routing

Using link packet drop probabilities obtained from (7.12), we can formulate the optimal general routing problem as:

$$\Phi = \sum_{l \in \mathcal{L}} \lambda_l P_l \tag{7.13}$$

where λ_l is the arrival rate for link l and P_l is its drop rate calculated by (7.12). This is a constrained optimization problem with the flow constraints at each router j for each demand $\mathcal{D}(s,t)$ between source s and destination t. If $f_l^{(s,t)}$ denotes the fraction of the demand $\mathcal{D}(s,t)$ on link l, then the flow balance constraints are given by

$$\sum_{i:(i,j)\in\mathcal{L}} f_{(i,j)}^{(s,t)} - \sum_{i:(j,i)\in\mathcal{L}} f_{(j,i)}^{(s,t)} = \begin{cases} -\mathcal{D}(s,t) & \text{if } j = s \\ \mathcal{D}(s,t) & \text{if } j = t \\ 0 & \text{Otherwise} \end{cases} \tag{7.14}$$

The mean packet arrival rate to a link l, λ_l, is given by

$$\lambda_l = \sum_{(s,t)\in\mathcal{N}\times\mathcal{N}} f_l^{(s,t)} \tag{7.15}$$

The parameter $p^{(s,t)}$ for the GE process used to fit the demand $\mathcal{D}(s,t)$ is given according to (7.7):

$$p^{(s,t)} = \frac{2\mathcal{D}(s,t)^2}{\mathcal{D}(s,t)^2 + \mathcal{V}(s,t)} \tag{7.16}$$

Let $r_l^{(s,t)}$ denote the probability with which the demand $\mathcal{D}(s,t)$ is sent on link l. Then $r_l^{(s,t)}$ is given by

$$r_l^{(s,t)} = \frac{f_l^{(s,t)}}{\mathcal{D}(s,t)} \tag{7.17}$$

Let $p_l^{(s,t)}$ denote the parameter p of the GE process after splitting the demand $\mathcal{D}(s,t)$ with probability $r_l^{(s,t)}$. Then $p_l^{(s,t)}$ denotes the parameter p of the GE process representing the flow $f_l^{(s,t)}$. The parameter $p_l^{(s,t)}$ is given according to (7.8):

$$p_l^{(s,t)} = \frac{p^{(s,t)}}{p^{(s,t)}(1 - r_l^{(s,t)}) + r_l^{(s,t)}} \tag{7.18}$$

The total offered load on link l is given by λ_l (7.15), the parameter p of the associated GE distribution may be obtained by merging the flows $f_l^{(s,t)}$ going through l. If p_l denotes the parameter p of the GE process associated with the aggregate traffic on link l, then p_l is given by

$$p_l = \lambda_l \Big(\sum_{(s,t) \in \mathcal{N} \times \mathcal{N}} f_l^{(s,t)} p_l^{(s,t)} \Big)^{-1} \tag{7.19}$$

If ρ_l is equal to $\frac{\lambda_l p_l \bar{X}}{B_l}$, then, using (7.12), the probability of packet dropped at link l is given by

$$P_l = \frac{(p_l - \rho_l)(\rho_l + 1 - p_l)^{K_l}}{1 - (\rho_l + 1 - p_l)^{K_l + 1}} \tag{7.20}$$

The optimal general routing problem is given by (7.13), subject to the constraints given by (7.15), (7.16), (7.17), (7.18), (7.19), (7.20). It may be noted that we are casting the traffic according to the routing in order to obtain the mean and variance of the total offered traffic to each $l \in \mathcal{L}$. However, we are not iterating to obtain the equilibrium traffic parameters. Essentially, we are using the upper bound on the packet drop probability in (7.13).

7.2.2 Optimization of OSPF Weights Using On-line Simulation

The general optimal routing problem, where the objective function is completely defined by (7.13)-(7.20), may possibly be solved for $f_l^{(s,t)} \forall l \in \mathcal{L}$ by using some non-linear programming techniques. However, under constraints of OSPF routing,

the relation between the link weights and optimization metric can no longer be analytically defined. Hence, the optimal routing in OSPF becomes a "black box" optimization problem which may be defined as:

$$\min \Phi(\mathbf{w}) \tag{7.21}$$

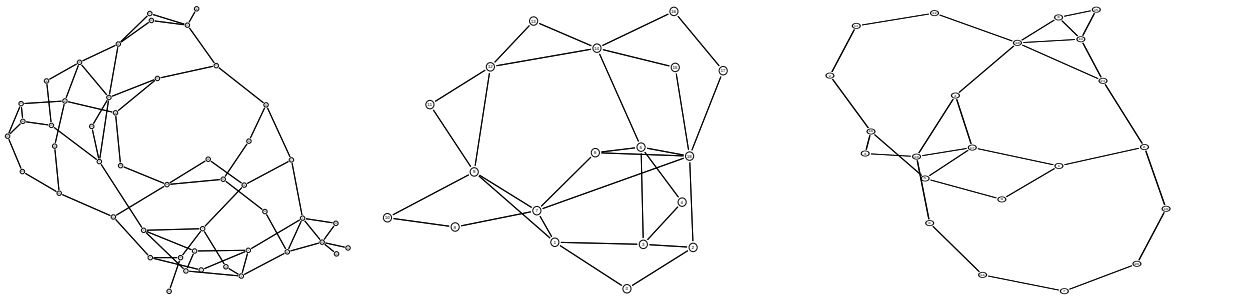
where $\mathbf{w}$ is the vector of network link weights and $\Phi(\cdot)$ the objective function, which is unknown. Basically, in order to obtain the value of Φ for a given OSPF weight setting, we run modified Floyd Warshall's algorithm (modified to obtain equal cost paths also) to obtain the routing. Then the traffic is cast to obtain parameters of the aggregate packet arrival process and drop probability for every link $l \in \mathcal{L}$ using (7.15), (7.16), (7.17), (7.18), (7.19) and (7.20). Finally the value of Φ may be calculated by (7.13). In [106], authors have proved that it is NP-hard to find OSPF link weight settings for an optimization metric piecewise linear in offered load. It is straightforward to show, by proceeding along the same lines, that our problem, i.e., minimize the packet drop rate given by (7.13) is also NP-hard. For such NP-hard problems, heuristic optimization algorithms are usually used to search for approximate solution. In this work, the on-line simulation scheme is used to dynamically optimize OSPF link weights.

7.2.3 Simulation Results

In this section we present two sets of simulation results. One is to demonstrate that the Recursive Random Search obtains better OSPF link weight settings with fewer function evaluations than the algorithm proposed in [28]. Another set of results demonstrate the improvement in end-to-end performance (in terms of the drop rate) by dynamic optimization of OSPF weights.

We have considered three network topologies, shown in Figure 7.9, to demonstrate our results. Two are well-known ARPANET topology and MCI topology. We couldn't include AT&T topology used in [28] since it is not publicly available. The ARPANET topology consists of 48 routers and 140 simplex links Each link in the network is assumed to consist of two simplex link whose weights may be set independently. MCI topology consists of 19 routers and 62 simplex links. We have also

considered a randomly generated topology with 22 routers and 60 simplex links.



(a) ARPANET Topology (b) MCI Topology (c) Randomly Generated Topology

Figure 7.9: Figure showing the network topologies used in simulation

Random amount of traffic was sent from every node to every other node in the network. This random traffic was generated using the method outlined in [28]. For each node u, two random numbers are generated O_u, $D_u \in [0, 1]$. For each pair of nodes (u, v) another random number $C_{(u,v)} \in [0, 1]$ was generated. If Δ denotes the largest Eucledian distance between any pair of nodes and if α denotes a constant, the average demand between u and v is given by

$$\mathcal{D}(u, v) = \alpha O_u D_v C_{(u,v)} e^{\frac{-\delta(u,v)}{2\Delta}}$$

where, $\delta(u, v)$ denotes the Eucledian distance between the nodes u and v. This method of generating random traffic (the term $e^{\frac{-\delta(u,v)}{2\Delta}}$) ensures more traffic for source destination pairs that are closer to each other. Since a product of three random variables is taken to generate the demands, there is actually a large variation in the traffic demands. The ratio of square of mean to the variance was assumed to be a uniformly distributed random variable in $[0, 1]$. The mean and variance of the traffic demands are generated using the above procedure. All the links in the network have 1Mbps bandwidth with a buffer size of 50 packets. The packet size was chosen to be exponentially distributed with mean packet size of 200 bytes.

In the simulation results, we do not verify the traffic modeling assumptions as this is not a focus of this work. The performance results shown in 7.2.3.1 are the average results from ten simulation runs. Average of multiple simulation runs is presented as we compare the performance of two stochastic search algorithms.

7.2.3.1 Comparison of Search Schemes

In this section, we present the results of comparison of Recursive Random Search with the local search scheme proposed in [28]. In optimization literatures, the comparison between algorithms is usually done in terms of the number of function evaluation instead of the absolute time taken to find a "good" parameter setting. This is because the computation time is considerably dependent on many other factors, such as, implementation efficiency, testing platform, etc.. Considering the main computation time is for function evaluations, the number of function evaluation is a more appropriate performance metric under the assumption that the computation time per function evaluation is approximately the same for both schemes. Note this assumption is not exactly true in the context of our problem, where one function evaluation represents one optimization metric computation for a specific set of link weights. In [28], authors have used incremental shortest path computations to improve the speed of search as very few link weights change from one iteration to the next which is reported to have 15% improvements on an average. In spite of this, we still use the number of function evaluations as our algorithm performance metric for the reasons mentioned above and the consideration that our algorithm is designed to be a general "black-box" search algorithm where no problem-specific is available. It should be noted that even if taking 15% improvement for the local search scheme of [106] into consideration, the test results still show that our algorithm is significantly faster.

Loosely, we refer to the number of function evaluations required to obtain a "good" parameter setting as the speed of convergence. A "good" parameter setting has been defined as the OSPF link weight setting that give metric value lower than that by setting all link weights equal to unity (called unit OSPF). This definition is just for the purpose of comparison. A "good" parameter setting may have been defined alternatively as the link weight setting to achieve performance metric equal to, say, 80% of the unit OSPF.

7.2.3.2 Heuristic Piecewise Linear Metric

In order to compare the speed of convergence of our search scheme with the local search scheme proposed in [28], we use the same metric used in [28], which is piecewise linear with the link offered load.

Figure 7.10 shows the optimization convergence curves for the ARPANET, MCI and Randomly generated network topologies respectively. For the sake of comparison, these graphs also show the optimization metric value when all the links' weights are set to unity. It can be seen that the recursive random search scheme outperforms the local search scheme in terms of the number of function evaluations needed to find a "good" parameter setting for all three network topologies. These results have been tabulated in Table 7.1.

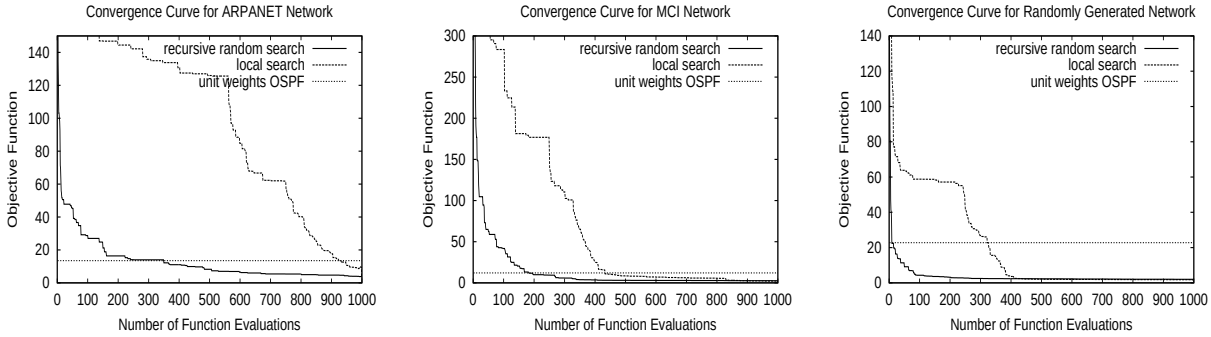


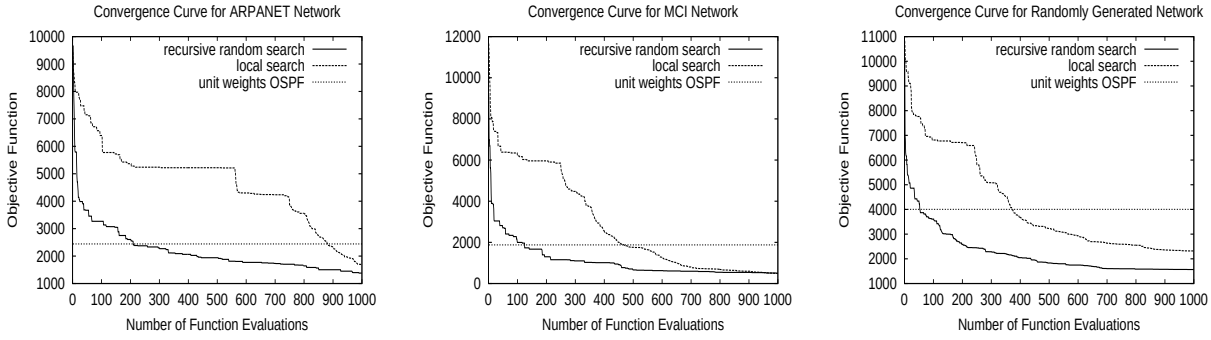
Figure 7.10: Figure showing the convergence curves of piecewise linear metric for (a) ARPANET (b) MCI (c) Randomly generated network topology

Scheme	ARPANET	MCI	Random
Local Search	932	433	322
RRS	350	183	9
Improvement	62.4%	57.7%	97.2%

Table 7.1: Table comparing the number of function evaluations needed to obtain a "good" parameter setting for piecewise linear metric

7.2.3.3 Packet Drop Rate Metric

In this section we present the comparative results for the packet drop metric defined in (7.20). Figure 7.11 shows the comparison results of the optimization



(a) (b) (c)

Figure 7.11: Figure showing the convergence curve of total packet drop rate for (a) ARPANET (b) MCI (c) Randomly generated network topology

convergence speed. The results clearly show that the recursive random algorithm significantly outperforms the local search algorithm. Table 7.2 shows that for the packet drop rate metric, our recursive random search scheme took 70% or fewer function evaluations to obtain a "good" OSPF link weight setting.

Scheme	ARPANET	MCI	Random
Local Search	882	469	372
RRS	210	125	54
Improvement	76.1%	73.3%	85.5%

Table 7.2: Table comparing the number of function evaluations needed to obtain a "good" parameter setting packet drop rate metric

7.2.3.4 Optimizing OSPF for Improving Packet Drop Rate

Now we describe the simulation showing how the network performance can be improved by our OSPF optimization scheme. Figure 7.12 shows the functional block diagram of the overall setup of this simulation. The OLS monitors the traffic to provide the estimates of mean and variance of the traffic demand for performance evaluation of link weights. Recursive random search is then be used to search for better link weight setting for the network. When a certain stopping criteria is met, for example, the time limit is reached, the best-so-far link weight setting found by RRS may be deployed in the real network if it results in substantial improvement in the performance.

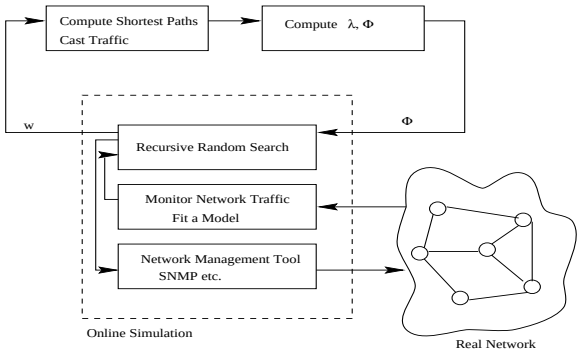
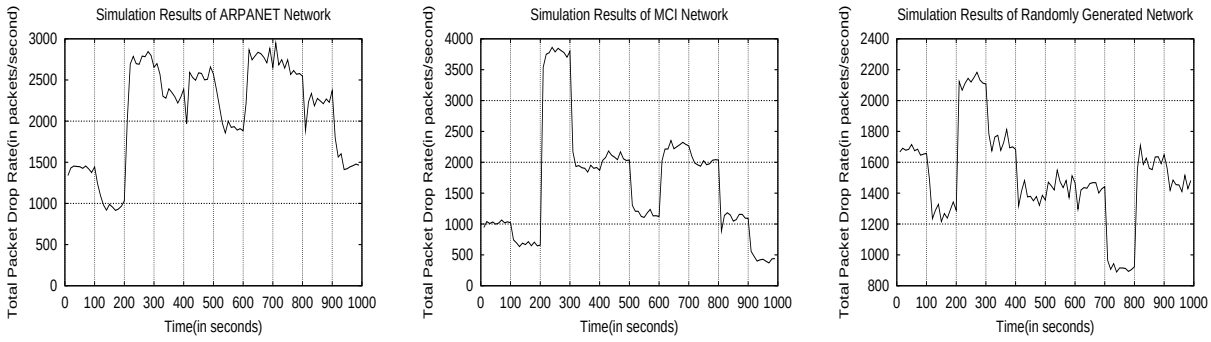


Figure 7.12: Overall OSPF optimization setup using on-line simulation architecture



(a) (b) (c)

Figure 7.13: Figure showing total packet drop rate as a function of time for the (a) ARPANET (b) MCI (c) Randomly generated network topology. Traffic pattern was changed at times 0, 200, 400..., the optimized OSPF weights were deployed at times 100, 300,...

We used *ns*[3] to simulate the real network running OSPF. The traffic in the network was generated in the same way as outlined in the beginning of this section. However, every 200 seconds the traffic pattern (the mean and variance of demand matrix) was changed in order to create a dynamic scenario. The traffic generator is implemented over UDP to generate bursty traffic with the GE inter-arrival distribution described in (7.6). In our simulation, we assume OLS has a complete knowledge of necessary network information, such as, traffic demands, network topology, etc.. Whenever a change of traffic pattern happens, the OLS runs the recursive random search for a certain iterations to obtain a better parameter setting. If the optimized setting is better than the original, it will be deployed at 100

seconds after the traffic change. The 100-seconds time difference is used because we want to observe the performance difference between before optimization and after optimization. Note that here we assume the running time of the search algorithm is faster than the traffic change period, i.e., the search algorithm has finished running at 100 seconds after the traffic change.

The actual packet drop rates are collected during the simulation for all the traffic sinks in the network and then summed together to get the total packet drop rate. Figure 7.13 shows total packet drop rate in the network as a function of time. Table 7.3 summarizes the maximum improvement in packet drop rates for different topologies. Note that more or less improvements may result depending on the topology and traffic conditions.

	ARPANET	MCI	Random
Max. Improvement	31.8%	60.2%	35.7%

Table 7.3: Table summarizing the maximum percentage improvement in the packet drop rates obtained for different topologies for the results shown in Figure 7.13

7.2.4 Conclusions

We have investigated the problems associated with the optimization of OSPF weights using on-line simulation framework. The optimization problem was formulated where total packet drop rate was chosen as the optimization criteria. The simulation results demonstrate that our search algorithm took 50-90% fewer function evaluations to find a good OSPF weights "setting" as compared to the local search algorithm of [28]. The simulation results of our OSPF optimization scheme also demonstrate improvements of the order of 30-60% in the total drop rate in the network.

7.3 Optimize BGP Routing Algorithm

Intra-domain traffic engineering can be performed with the global knowledge of network-wide traffic demands since all the routers are under the same administrative organization and the control over the routers in the network is accessible.

However, in the case of inter-domain TE involving different administrative organizations, the traffic demand statistics are usually kept private and the control over routers outside the administrative organization is normally not practical. Therefore, inter-domain TE has mainly focused on multi-homed Autonomous Systems (AS), in-bound/out-bound load-balancing between adjacent ASes using BGP attributes (e.g. `MED`, `local_pref`, `as_path`, etc.) [117].

The ASes are increasingly becoming multi-homed [117]. The outbound traffic of an AS may be routed on any of the multiple links, depending on the decision made by the inter-AS routing algorithm, usually Border Gateway Protocol(BGP). BGP routing decisions are made by a series of policy filters. Most ASes use the shortest AS path for most destinations. This may lead to unbalanced load even amongst the multiple outbound interfaces of an AS. In this section, we consider the problem of load-balancing outbound traffic in BGP from the perspective of a single AS. We show that this is an NP-hard problem and use the OLS tool to solve this problem.

BGP provides only some simple capabilities for TE between AS neighbors. The `MED` attribute can be used by an AS to inform its neighbor of a preferred connection (among multiple physical connections) for inbound traffic to a particular address prefix. Usually it is used by the service providers on the request of their multi-homed customers. Lately, it is also being used between the service providers. The `as_path` attribute has also been used to achieve TE objectives. `as_path` is "stuffed" or "padded" with additional instances of the same AS number to increase its length and expect lower amount of inbound traffic from the neighbor AS to whom it is announced. However, this may lead to a large overhead if done too often. Another way used to achieve some TE is to subvert the BGP-CIDR address aggregation process. In particular an AS may extract more-specifics, or de-aggregate it and re-advertise the more-specifics to other ASes. The longest-prefix match rule in IP forwarding will lead to a different route for the more specific address. However, this is achieved at the expense of larger number of entries in forwarding tables. This is an indirect and undesirable way to achieve inbound load-balancing. One way to avoid subverting CIDR aggregation (shown in our recent work [118]), in the case of multi-

homed *stub AS*, is by mapping the inbound load-balancing problem to an address management problem. Alternatively, AS neighbors may agree on BGP community attributes [119] (that are not re-advertised) to specify traffic engineering. We notice that inbound load-balancing is considerably complex and requires re-advertisements or support from neighboring ASes. However, outbound load-balancing is simpler, and can be achieved by impacting local policy changes.

The `local_pref` attribute is used locally within the AS to prefer an outbound direction for a chosen destination prefix, AS or exit router. `local_pref` holds the highest priority in the policy filter hierarchy, i.e. the BGP will choose the path with highest `local_pref` over other policy attributes. Therefore, if we know the desired routing to meet the traffic engineering objective, we can use the `local_pref` to over-ride the default routing. Recent work [120] notice that it is possible to automatically tune the `local_pref` parameters of "hot-prefixes" to control outbound traffic subject to a range of policy constraints. However, they do not provide any mechanism to do this. In this section, we use the on-line simulation scheme to calculate the ideal routing based on the prevailing traffic conditions and deploys the preferred routing by adjusting `local_pref` attributes.

7.3.1 Traffic Demands

Given a certain outbound traffic demand, load balancing aims to split this traffic demand and distribute them evenly among outbound links. Usually, the traffic demand can be divided into a number of traffic flows. In the finest granularity, a traffic flow is determined by the source and destination IP addresses and the port number. In a coarse granularity, a traffic flow can be identified by the source and destination AS-pair. Internet measurements have shown that traffic aggregates based on destination prefixes in the routing table are more suitable for load balancing [103] since they are relatively stable through the day and on per-hour time scales. We have used this granularity for defining a flow in our load balancing scheme. In other words, the traffic demand is split into flows at the level of per destination-prefix.

A typical BGP routing table consists of thousands of destination prefix entries. It will be very complex to work with such a large number of traffic flows. However,

many traffic measurements [103, 104] have demonstrated the existence of so-called elephant and mice phenomenon. That is, a small number of traffic streams, known as *elephants*, generate a large portion of total traffic whereas a large number of streams, *mice*, generate a small portion of total traffic. For example, it has been found that the top 9% of flows between ASes account for 86.7% of the packets or 90.7% of the bytes transmitted [104]. Furthermore, these elephant traffic flows are usually very stable over time and hence are suitable to be re-routed for load-balancing purpose. Based on these observations, our load balancing scheme only attempt to adjust the routing of the top 10% destination prefixes in the routing table based on their traffic demands. [1].

7.3.2 Optimal Routing Calculation for Load Balancing

Given the knowledge of traffic demand and outbound link information, the optimal routing for load balancing can be calculated. Let m be the number of outbound links in the concerned AS. Let l_i and c_i, $i = 1 \ldots m$, denote the i^{th} outbound link and its capacity (or bandwidth), respectively. All the outbound traffic of this AS will be routed on these links. If s_i, $i = 1 \ldots m$, denotes the total outbound traffic carried by the i^{th} link, then the utilization of link l_i is given by s_i/c_i. The objective of load balancing is to minimize the maximum link utilization among all the outbound links, i.e.,

$$\text{minimize} \max_{i=1 \ldots m} \frac{s_i}{c_i} \tag{7.22}$$

Let n denote the number of selected destination prefixes and d_j, $j = 1 \ldots n$, denote the average offered load for these destinations. Our load balancing scheme attempts to adjust the routing of these n prefixes in order to minimize the objective function in Equation (7.22). Let $\mathcal{D}_i$ denote the subset of the n prefixes that are routed on link l_i under adjusted routing. If f_i denotes the load on link l_i generated by the other 90% traffic flows, which is sent on this link by the default BGP routing,

[1]The fraction of optimized destination prefixes can be kept fixed or increased in the event of increase in routing tables. In future, a smaller fraction of destination prefixes may be used if 10% gives a very large number.

then Equation (7.22) becomes

$$\text{minimize } \Phi \;=\; \max_{i=1\ldots m}\Big(\sum_{j\in\mathcal{D}_i}\frac{d_j}{c_i}\Big)+\frac{f_i}{c_i} \tag{7.23}$$

where, the first term represents the percentage load due to the selected 10% flows and the second term represents the percentage load generated by the other 90% flows on link l_i. This problem can also be written as the following integer programming problem.

$$\begin{aligned} \text{minimize} \quad & t \\ \text{subject to} \quad & \sum_{j=1}^{n} x_{ij}\frac{d_j}{c_i}+\frac{f_i}{c_i}\le t,\ i=1\ldots m \\ & \sum_{i=1}^{m} x_{ij}=1,\ j=1\ldots n \\ & x_{ij}\in\{0,1\},\ i=1\ldots m,\ j=1\ldots n \end{aligned} \tag{7.24}$$

where x_{ij} is a binary number and $x_{ij}=1$ means flow d_j is output on link l_i, otherwise $x_{ij}=0$. Note that traffic flow d_j may not have all outbound links as its alternative paths. One can assume an arbitrarily large d_j/c_i for those links. The problem represented by Equation (7.24) is actually a classical task scheduling problem with unrelated parallel machines [121], where a number of tasks with different sizes are assigned to a set of parallel machines. The processing time of each task is different on different machines and the objective there is to minimize the completion time of all tasks by carefully distributing these tasks onto the parallel machines. This problem is NP-hard and approximation algorithms can be used to obtain approximate solutions. For example, in [122] a linear programming technique is first used to obtain a basic solution where there are at most $m-1$ non-integral x_{ij}. Then for these non-integral x_{ij}, an exhaustive enumeration is performed to find the optimal scheduling. Combining the solutions of these two steps can produce an approximate solution with a upper bound of $2t^*$, where t^* denotes the value of t produced by the optimal solution. The time complexity of this method is exponential in the value of m.

In stead of the integer programming approach, we have applied the OLS scheme to this load balancing problem. With the OLS scheme, it is also possible to optimize for various performance objectives in addition load balancing, or multiple objectives by formulating appropriate objective functions and combining these objectives using multi-objective optimization techniques. For example, in addition to load-balancing, the network operator also prefers to use the shortest paths. It is possible to formulate a multi-objective optimization problem and obtain a solution, using OLS, that meets both load-balancing and shortest path criteria.

7.3.3 The BGP Optimization Scheme

The complete procedure can be summarized as follows:

Step 1 Extract top 10% destination prefixes, with traffic demands $d_j,\ j = 1 \ldots n$, from the routing table;

Step 2 Calculate d_j/c_i and $f_i/c_i,\ i = 1 \ldots m,\ j = 1 \ldots n$ according to the traffic demand for each prefix and the capacity of each outbound link.

Step 3 Each destination prefix may be reachable by all or some of the outbound links. This information can be obtained from Adj-RIBs-In at a BGP router. Assign a very large value of d_j/c_i for the infeasible routes, so the solution (minimization) will not result in an infeasible solution.

Step 4 Measure or compute the value of Φ for default routing using Equation (7.23) denoted by Φ^0.

Step 5 Obtain a routing $\overline{r}$ from the RRS; Run RRS till a stopping criteria is reached. A stopping criteria can be a limit on time, number of iterations etc.. Let Φ^*, $\overline{r}^*$ denote the value of objective function and corresponding routing at the end of optimization.

Step 6 If $|\frac{\Phi^0 - \Phi^*}{\Phi^0}| \geq \Delta$, deploy $\overline{r}^*$ by setting a high `local_pref` of desired links for appropriate destination prefixes.

7.3.4 Simulation Results

We have used two optimization objectives, load-balancing objective given by Equation (7.22) and the minimization of packets dropped, to illustrate our results. We have use the ology shown in Fig-

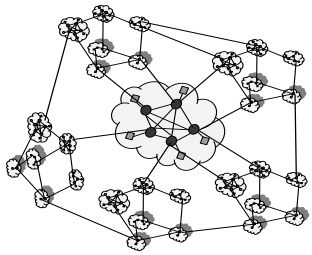


Figure 7.14: Network topology used for simulation results

ure 7.14. This network consists of 31 ASes, 90 routers and 90 hosts. In the simulation, the ASes did not have any policies (apart from the `local_pref` used to deploy the optimized routing). We optimize the performance for the AS 0 (shown in the center of graph).

7.3.4.1 Optimizing for Load Balancing

For the concerned AS, there are 8 outbound links with normalized capacity of 100, 100, 100, 100, 45, 45, 45, 12, respectively. We assume the number of top 10% destination prefixes which generate most of the traffic is 148, i.e., there are about 1480 prefixes in the routing table. Note this number is chosen somewhat arbitrarily only for the sake of illustration. In the simulation, we generate only 148 traffic flows instead of all the traffic flows since the actual effect of all the other 90% flows on the simulation is only to reduce the capacity of the links by a certain amount. Therefore, ignoring these flows will not compromise the validity of the simulation results in any way. We assign each destination prefix a certain load such that the total offered load is the 30% of the total capacity of all the links. In the beginning of the simulation, the routing of outbound traffic is decided by the default BGP routing. Then in the simulation, we apply the proposed load balancing scheme to the network. The maximum link utilization (given by Equation 7.22) before and

after optimization is shown is Table 7.4 and the link utilization of outbound links is compared in Fig 7.15.

	Before optimization	After optimization
Max. Link Utilization	91%	35%

Table 7.4: Maximum link utilization before and after optimization

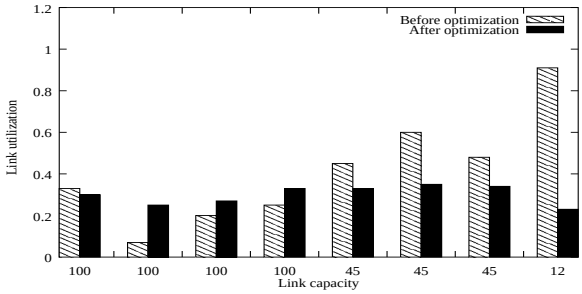


Figure 7.15: Link utilization of different outbound links before and after optimization

Figure 7.15 shows that the default BGP routing leads to an uneven distribution of load across the outbound links, for example, one link is greatly under-utilized with a utilization of 7% while one other link is approaching full utilization with a utilization of 0.91%. After optimization with the proposed scheme, the load distribution become much more even over the outbound links and the utilization of all the links is very close to the ideal value, i.e., the average utilization 30%. This simulation demonstrates that using the proposed optimization approach, the load balancing on the outbound links can be effectively achieved. Note that the optimization result shown in the table is obtained with an optimization process of only 1500 function evaluations. In a 500MHz Pentium PC, it only takes around 1 minute to finish the optimization process. Therefore, this optimization process can be used for on-line tuning of the load balancing objective.

7.3.4.2 Minimizing Packet Loss

In this section, we present the simulation results by using the packet loss objective. We used the same topology (shown in Figure 7.14) for the results presented in this section. However, the bandwidth of all the links was assumed to be 10Mbps.

During the simulation, the number of packets dropped on the outbound links is collected and the overall packet drop probability is calculated by the total packet number arrived on all outbound links. The optimization objective here is to minimize the overall packet drop probability. Table 7.5 shows the average packet drop probability before and optimization optimization and Figure 7.16 compares the number of packets dropped on each outbound link. We see that the average packet drop

	Before optimization	After optimization
% Packet Drop	41.07%	6.74%

Table 7.5: Comparison of average packet loss probability at outbound links before and after optimization

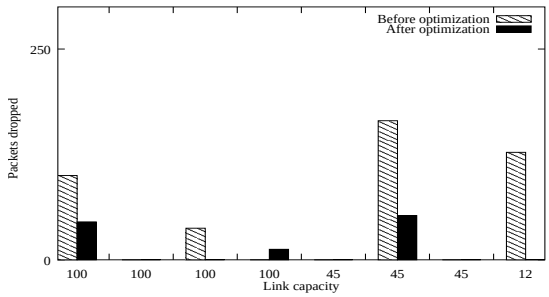


Figure 7.16: Packets dropped on various outbound interfaces before and after optimization

probability of the original routing is very high because of the uneven load distribution. After optimization, the average packet drop probability reduces dramatically since no link is over utilized.

CHAPTER 8
Conclusion and Future Research Direction

8.1 Conclusion

Network performance management can be accomplished with a black-box optimization approach, where the network is considered as a black box with a set of parameters, network simulation is used to evaluate the performance of the black box and an optimization algorithm is employed to optimize the parameters for better performance. Based on this approach, an on-line simulation scheme has been proposed as a general network performance management framework. This dissertation investigated the problem of designing an efficient optimization algorithm for such a on-line simulation scheme and validated the effectiveness of the proposed scheme with simulations and experiments on some important network protocols.

Due to No Free Lunch theorem, no optimization algorithm can perform consistently better than the others for all classes of problems. The "best" algorithm is only locally meaningful, i.e., one algorithm may outperforms the others for a certain class of problems, but its performance will suffer in other classes of problem. The average performance of any optimization algorithm over all classes of problems is the same. Therefore a trade-off between applicability and efficiency has to be made in practice. An optimization algorithm claiming to be efficient always has more restrictions on applicable problems while a widely applicable algorithm usually means an average performance. Therefore, to design a good optimization algorithm is effectively to find a proper balance between applicability and efficiency.

This dissertation first examined the general requirements of network optimization problems, i.e., high efficiency, scalability to high-dimensions, robustness to noises in objective functions. Based on these requirements, the Recursive Random Search algorithm has been designed to perform efficient search for concerned network optimization problems. According to the design objective, the emphasis of RRS is not on "full" optimization but on quickly finding near-optimum solutions within the limited time frame. Furthermore, the proposed algorithm is not claimed

to be the most efficient algorithm for every network optimization problem, instead, the algorithm is intended to be a general solution for the network optimization problems, which aim to achieve a good balance between efficiency and applicability. In other words, RRS tries the best to perform efficiently and consistently for various network optimization problems.

To achieve high efficiency for different problems, proper search techniques have to be selected to exploit the structural properties of the underlying problem. In addition, a search algorithm for practical optimization problems should also try to fully utilize the available computing resource, which may have parallel computing capability. To address these issues, the Unified Search Framework has been designed, which integrates various search techniques and selects appropriate ones based on the features of the underlying problem. In addition, the USF also employs a unified resource management mechanism to fully exploit available computing resources and achieve the maximum efficiency.

The adaptive network performance management using on-line simulation scheme has been validated with simulations and experiments on the optimization of three important network protocols: RED queueing management, OSPF intro-domain routing and BGP inter-domain routing. The problem formulation of optimizing the concerned protocols are first studied. A network protocol is employed to achieve one or multiple objectives which may correlated with each other. To formulate the optimal configure of this protocol as an optimization problem, a single optimization metric has to be carefully designed to quantify the concerned objectives so that the optimization of this metric will lead to the desired results. A poorly designed metric, which does not reflect the objectives controlled by the underlying network protocol, will fail the optimization scheme. This dissertation carefully formulated the optimization problems of the concerned network protocols. The results from simulations and experiments have demonstrated that the proposed approaches can significantly improve network performance under varying network conditions and the RRS algorithm performs very efficiently in these applications.

8.2 Future Research Direction

Many engineering design problems can be formulated as black-box optimization problems. The optimization strategies proposed in this dissertation not only can be used in the optimization of network protocols as described, they can also be used in many other situations. In fact, a wide range of optimization problems have similar features as network optimization problems. That is, these problems emphasize efficiency more than full optimization, and the objective function is evaluated with simulation and noises may be introduced into the function evaluation by inaccuracies in simulation. Recursive Random Search algorithm can be used to solve these problems efficiently. Furthermore, the Unified Search Framework is more widely applicable since it includes various search techniques. The USF can be used as a general solution for any kind of black-box optimization problems.

For a black-box optimization problem, since little knowledge is available to exploit, random sampling seems the best one can do. To perform better than random sampling, an optimization algorithm has to extract the structural properties of the underlying problem and exploit them during its optimization process. Therefore, the optimization process can be considered as a learning process, during which the structural properties are found through sampling and then used to expedite the optimization. This learning process is crucial to the efficiency of an optimization algorithm. Statistical analysis is a powerful technique to extract information from samples and can be used in the optimization algorithm. Recursive Random Search uses the statistics of the function values to guide the further optimization and is one attempt to apply statistical techniques in optimization process. Many aspect of RRS may be enhanced by using other statistical techniques. For example, in RRS, when the sample space is re-aligned around the center of a new sample, the size of sample space can also be adjusted by estimating the tail distribution of the function value of the center point value since this tail distribution is just the Lebesgue measure of all points with better function values than the center sample point. Another possible way to further refine the sample space is to use the statistics of multiple samples, such as, the ranges of their parameters, to decide the location and the shape of new sample space instead of a simple hypercube when using a single sample.

Since there is no single all-purpose optimization algorithm which is efficient for all problems, the essential task in a practical optimization problem is to choose appropriate search techniques based on the features of the given problem. Currently, this is typically achieved by the practitioner's experience or trial and error. The Unified Search Framework has been proposed in this dissertation as a general platform to implement this ideas. In USF, the correlation between structural properties of the underlying problem and search techniques has been investigated. Based on this, one can select appropriate search techniques as building blocks to build a tailed optimization algorithm for the underlying problem. It is more desirable that USF can automatically analyze the underlying problems and come up with a proper selection of search techniques. It is essentially a decision process in which a number of samples are used to decide if a certain structure exists. This is a very challenging problem and there has been not much work done in this area. Statistical techniques might be a solution to this problem. Basically, an appropriate statistics should be designed to characterize the features of the samples. For example, with an objective function with many separable parameters, the function values of random samples should follow a Gaussian distribution. Therefore, we can make approximate inference about the separability of the objective function by applying hypothesis testing on the distribution of function values of random samples. Note this example is just intended to demonstrate the idea. Actually Gaussian distribution of objective function values does not mean parameter separability. The objective function without parameter separability may also have Gaussian distributed random samples. In fact, the design of an appropriate statistics is the most important task in applying statistical techniques to structural property identification.

LITERATURE CITED

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

[2] Tao Ye and et al. Traffic management and network control using collaborative on-line simulation. In *Proc. of IEEE ICC'01*, Helsinki, Finland, 2001.

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

[4] SSFNET. network simulator. http://www.ssfnet.org.

[5] GloMoSim. network simulator. http://pcl.cs.ucla.edu/projects/glomosim.

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

[7] D. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. MA: Addison Wesley, 1989.

[8] E. Aarts and J. Korst. *Simulated Annealing and Boltzmann Machines*. John Wiley & Sons, 1989.

[9] 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.

[10] A. Törn, M.M. Ali, and S. Viitanen. Stochastic global optimization: Problem classes and solution techniques. *Journal of Global Optimization*, 14:437–447, 1999.

[11] Nicholas J. Radcliffe and Patrick D. Surry. Fundamental limitations on search algorithms: Evolutionary computing in perspective. In *Computer Science Today*, pages 275–291. 1995.

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

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

[14] W. Zhang and T. G. Dietterich. A reinforcement learning approach to job-shop scheduling. In *Proceedings of the International Joint Conference on Artificial Intellience*, 1995.

[15] M. R. Hoare and P. Pal. Physical cluster mechanics, statics and energy surfaces for monatomic systems. *Adv. Phys.*, 20:161–196, 1971.

[16] A Neumaier. Molecular modeling of proteins and mathematical prediction of protein structure. *SIAM Review*, 39:407–460, 1997.

[17] OSI system management overview. ISO-10040.

[18] William Stallings. *SNMP, SNMPv2, and CMIP: The Practical Guide to Network Management Standards*. Addison-Wesley, 1993.

[19] J. Case, M. Fedor, M. Schoffstall, and J. Davin. A simple network management protocol. RFC 1157, 1990.

[20] Principles for a telecommunications management network. ITU-T Rec. M.3010, 1992.

[21] J. Case SNMPv2 Working Group, K. McCloghrie, M. Rose, and S. Waldbusser. Protocol operations for version 2 of the simple network management protocol (snmpv2). RFC 1905, 1996.

[22] J. Case, R. Mundy, D. Partain, and B. Stewart. Introduction to version 3 of the internet-standard network management framework. RFC 2570, 1999.

[23] S. Waldbusser. Remote network monitoring management information base. RFC 1757, 1995.

[24] S. Waldbusser. Remote network monitoring management information base version 2. RFC 2021, 1997.

[25] R. Elbaum and M. Sidi. Topological design of local area networks using genetic algorithms. In *Proceedings of INFOCOM 1995*, 1995.

[26] B. Dengiz, F. Altiparmak, and A. Smith. Genetic algorithm design of networks considering all-terminal reliability. In *Proceedings of the 6th Industrial Engineering Research Conference*, pages 30–35, 1997.

[27] A.Pitsillides, G.Stylianou, C.S.Pattichis, A.Sekercioglu, and T.Vassilakos. Bandwidth allocation for virtual paths(bavp): Investigation of performance of classical constrained and genetic algorithm based optimization techniques. Tel Aviv,Israel, 2000.

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

[29] J. E. Falk and R. M. Soland. An algorithm for separable nonconvex programming problems. *Management Science*, 15(9):550–569, May 1969.

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

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

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

[33] R. Y. Rubinstein. *Simulation and the Monte Carlo Method*. John Wiley & Sons, New York, 1981.

[34] 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.

[35] J. Mockus. On bayesian methods of optimization. In L. C. W. Dixon and G. P. Szegö, editors, *Towards Global Optimization*, pages 166–181. North-Holland, 1975.

[36] M. J. Kushner. A new method of locating the maximum point of an arbitrary multipeak curve in the presence of noise. *Journal of Basic Engineering*, 86:97–106, 1964.

[37] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical Recipes in C*. Cambridge University Press, 2 edition, 1992.

[38] 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).

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

[40] 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.

[41] R. M. Lewis, V. Torczon, and M. W. Trosset. Direct search methods: Then and now. *Journal of Computational and Applied Mathmatics*, 124:191–207, December 2000.

[42] W. Spendley, G. R. Hext, and F. R. Himworth. Sequential application of simplex designs in optimization and evolutionary operation. *Technometrics*, 4:441–461, 1962.

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

[44] J. Nocedal. Theory of algorithms for unconstrained optimization. In A. Iserles, editor, *Acta Numerical*, pages 199–242. Cambridge University Press, Cambridge, 1992.

[45] M. J. D. Powell. A direct search search optimization method that models the objective and constraint functions by linear interpolation. Numerical Analysis Report, DAMTP 1992/NA5, Department of Applied Mathematics and Theoretical Physics, University of Cambridge, England, 1992.

[46] V. Torczon. *Multi-Directional Search: A Direct Search Algorithm for Parallel Machines*. PhD thesis, Department of Mathematical Sciences, Rice University, Houston, Texas, 1989.

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

[48] V. Torczon. On the convergence of pattern search algorithm. *SIAM Journal on Optimization*, 7:1–25, 1997.

[49] G. E. P. Box. Evolutionary operation: A method for increasing industrial productivity. *Appl. Statist.*, 6:81–101, 1957.

[50] 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.

[51] 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.

[52] 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.

[53] 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.

[54] Z. Michalewicz. *Genetic Algorithm +Data Structures = Evolution Programs.* Springer-Verlag: New York, 1996.

[55] A. H. Kan and G. T. Timmer. Stochastic global optimization methods part II: Multi level methods. *Mathematical Programming*, 39:57–78, 1987.

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

[57] M. Ali, A. Törn, and S. Viitanen. A numerical comparison of some modified controlled random search algorithms. *Journal of Global Optimization*, 11(4):377–385, 1997.

[58] I. Garcia, P.M. Ortigosa, L.G. Casado, G.T. Herman, and S. Matej. Multidimensional optimization in image reconstruction from projections. In *Developments in Global Optimization*, pages 289–300. Kluwer, 1997.

[59] Martin Pelikan, David E. Goldberg, and Fernando Lobo. A survey of optimization by building and using probabilistic models. In *Computational Optimization and Applications*. Kluwer Academic Publishers (in press, 2000.

[60] Melanie Mitchell, John Holland, and Stephanie Forrest. When will a genetic algorithm outperform hill climbing? In J. Cowan, G. Tesauro, and J. Alspector, editors, *Advances in Neural Information Processing Systems*. Morgan Kauffman, San Francisco, CA, 1994.

[61] M. Mitchell, S. Forrest, and J. H. Holland. The royal road for genetic algorithms: Fitness landscapes and ga performance. In F. J. Varela and P. Bourgine, editors, *Proceesings of the First Europen Conference on Artificial Life*, pages 245–254, Cambridge, MA, 1992. MIT Press.

[62] Anton Dekkers and Emile Aarts. Global optimization and simulated annealing. *Mathmatical Programming*, 50:367–393, 1991.

[63] H. Szu and R. Hartley. Fast simulated annealing. *Phys. Lett. A*, 122(3-4):157–162, 1987.

[64] L. Ingber. Very fast simulated re-annealing. *J. Mathl. Comput. Modelling*, 12:967–973,, 1989.

[65] Fred Glover. Tabu search– part I. *ORSA Journal on Computing*, 1(3):190–206, 1989.

[66] Fred Glover. Tabu search– part II. *ORSA Journal on Computing*, 2:4–32, 1990.

[67] Nanfang Hu. Tabu search method with random moves for globally optimal design. *International Joural for Numerical Methods in Engineering*, 35:1055–1070, 1992.

[68] R. Battiti and G. Tecchiolli. The reactive tabu search. *ORSA Journal on Computing*, 6(2):126–140, 1994.

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

[70] B. C. Arnold, N. Balakrishnan, and H. N. Nagaraja. *A First Course in Order Statistics*. John Wiley & Sons, 1992.

[71] 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.

[72] O. C. Martin and S. W. Otto. Combining simulated annealing with local search heuristics. Technical Report CS/E 94-016, Oregon Graduate Institute Department of Computer Science and Engineering, 1994.

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

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

[75] Zelda B. Zabinsky. Introduction to analyses of adaptive stochastic search methods for global optimization. Tutorial in INFORMS meeting 2001, 2001.

[76] M. F. Hussain and K. S. Al-Sultan. A hybrid genetic algorithm for nonconvex function minimization. *Journal of Global Optimization*, 11:313–324, 1997.

[77] Rainer Storn and Kenneth Price. Differential evolution-a simple and efficient heuristic for global optimization over continuousspaces. *Journal of Global Optimizaiton*, 11:341–359, 1997.

[78] R. Desai and R. Patil. Salo: combining simulated annealing and local optimization for efficient global optimization. In J.H. Stewman, editor, *Proceedings of the 9th Florida AI Research Symposium (FLAIRS-'96)*, pages 233–237, St. Petersburg, FL, USA, 1996.

[79] Donald R. Jones, Matthias Schonlau, and William J. Welch. Efficient global optimization of expensive black-box functions. *Journal of Global Optimizaiton*, 13:455–492, 1998.

[80] C. Mohan and K. Shanker. A numerical study of some modified versions of controlled random search method for global optimization. *International Journal of Computer Mathematics*, 23:325–341, 1988.

[81] M. Ali and C. Storey. Modified controlled random search algorithms. *International Journal of Computer Mathematics*, 53:229–235, 1994.

[82] Eligius M.T. Hendrix, P.M. Ortigosa, and I. Garcia. On success rates for controlled random search. *Journal of Global Optimizaiton*, 21:239–263, 2001.

[83] G. R. Wood, Z. B. Zabinsky, and B. P. Kristinsdottir. Hesitant adaptive search: The distribution of the number of iterations to convergence. *Mathematical Programming*, 89(3):479–486, 2001.

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

[85] 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.

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

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

[88] Ko-Hsin Liang, Xin Yao, and Charles Newton. Combining landscape approximation and local search.

[89] David E. Goldberg and Siegfried Voessner. Optimizing global-local search hybrids. IlliGAL Report 99001, Department of General Engineering, University of Illinois at Urbana-Champaign, January 1999.

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

[91] H. Mühlenbein, M. Schomisch, and J. Born. The parallel genetic algorithm as function optimizer. *Parallel Computing*, 17:619–632, 1991.

[92] J. E. Dennis, Jr. and V. Torczon. Direct search methods on parallel machines. *SIAM Journal of Optimization*, 1(4):448–474, 1991.

[93] G. Rudolph. Parallel approaches to stochastic global optimization. pages 256–267. IOS Press, Amsterdam, 1992.

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

[95] B. Braden and et al. Recommendations on queue management and congestion avoidance in the internet. RFC 2309, 1998.

[96] Wu chang Feng, Dilip D. Kandlur, Debanjan Saha, and Kang G. Shi. A self-configuring RED gateway. *Proceedings of IEEE Infocom 1999*, March 1999.

[97] Victor Firoiu and Marty Borden. A study of active queue management for congestion control. In *INFOCOM (3)*, pages 1435–1444, 2000.

[98] M. May, J. Bolot, C. Diot, and B. Lyles. Reasons not to deploy RED. Technical report, INRIA Sophia-Antipolis, France, 1999.

[99] M. Christiansen, K. Jeffay, D. Ott, and F.D. Smith. Tuning RED for web traffic. In *Proceeding of ACM SIGCOMM*, 2000.

[100] Thomas Bonald, Martin May, and Jean-Chrysostome Bolot. Analytic evaluation of RED performance. *IEEE Infocom 2000*, pages 1415–1444, 2000.

[101] C.V. Hollot, Vishal Misra, Don Towsley, and Wei-Bo Gong. A control theoretic analysis of red. In *Proceedings of IEEE Infocom 2001*, 2001.

[102] Sally Floyd, Ramakrishna Gummadi, and Scott Shenker. Adaptive RED: An algorithm for increasing the robustness of RED's active queue management. unpubished, 2001.

[103] 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.

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

[105] J. Moy. Ospf version 2. RFC 2178, April 1998.

[106] Bernard Fortz and Mikkel Thorup. Increasing internet capacity using local search. *IEEE Transaction on Networking*, 2000.

[107] Atul Khanna and John Zinky. The revised arpanet routing metric. In *Proceedings of the ACM SIGCOMM*, pages 45–56, 1989.

[108] David W. Glazer and Carl Tropper. A new metric for dynamic routing algorithms. *IEEE Transactions on Communications*, 38(3), March 1990.

[109] A. Sakamoto D. S. Lee, G. Ramamurthy and B. Sengupta. Performance analysis of a threshold-based dynamic routing algorithm. In *Proceedings of the Fourteenth International Teletraffic Congress*, 1994.

[110] Dimitri P. Bertsekas. Dynamic models of shortest path routing algorithms for communication networks with multiple destinations. In *Proceedings of 1979 IEEE Conference on Decision and Control*, pages 127–133, Ft. Lauderdale, FL, Dec 1979.

[111] Jon Crowcroft Zheng Wang. Analysis of shortest-path routing algorithms in a dynamic network environment. *Computer Communication Review*, 22(2):63–71, 1992.

[112] A. Feldmann, A. Greenberg, C. Lund, N. Reingold, J. Rexford, and F. True. Deriving traffic demands for operational ip networks: methodology and experience. *IEEE/ACM Transaction on Networking*, pages 265–278, June 2001.

[113] A. Feldmann, A. Greenberg, C. Lund, N. Reingold, and J. Rexford. Netscope: traffic engineering for ip networks. *IEEE Network Magazine: special issue on Internet traffic engineeringh*, pages 11–19, March/April 2000.

[114] Ramesh Nagarajan, James F. Kurose, and Don Towsley. Approximation techniques for computing packet loss in finite-buffered voice multiplexers. *IEEE J.Select.Areas Commun*, 9(3):368–337, April 1991.

[115] Robert B. Cooper. *Introduction to Queueing Theory*. New York : North Holland, second edition, 1981.

[116] Harry G. Perros. *Queueing Networks With Blocking, Exact and Approximate Solutions*. Oxford University Press, 1994.

[117] G. Huston. Commentary on inter-domain routing in the internet. RFC 3221, December 2001.

[118] T. Ye, S. Yadav, M. Doshi, A. Gandhi, S. Kalyanaraman, and H. T. Kaur. Load balancing in bgp environment using online simulation and dynamic nat. ISMA Workshop by CAIDA, December 2001.

[119] J. Stewart III. *BGP-4 Inter-domain routing in the Internet.* Addison-Wesley, 1999.

[120] Jay Borkenhagen, Nick Feamster, and Jennifer Rexford. Controlling the impact of BGP policy changes on IP traffic. NANOG, June 2002.

[121] L. A. Hall. Approximation algorithms for scheduling. In D. S. Hochbaum, editor, *Approximation Algorithms for NP-hard Problems.* PWS Publishing Company, 1995.

[122] C. N. Potts. Analyssis of a linear programming heuristic for scheduling unrelated parallel machines. *Discrete Appl. Math.*, 10:155–164, 1985.