

University of Nevada, Reno

Probabilistic Trans-Algorithmic Search

A dissertation submitted in partial fulfillment of the
requirements for the degree of Doctor of Philosophy in
Computer Science and Engineering

by

Bilal Gonen

Dr. Murat Yuksel
Dissertation Advisor

August, 2011



University of Nevada, Reno
Statewide • Worldwide

THE GRADUATE SCHOOL

We recommend that the dissertation
prepared under our supervision by

BILAL GONEN

entitled

Probabilistic Trans-Algorithmic Search

be accepted in partial fulfillment of the
requirements for the degree of

DOCTOR OF PHILOSOPHY

Murat Yuksel, Ph.D., Advisor

Bobby Bryant, Ph.D., Committee Member

Sushil Louis, Ph.D., Committee Member

Mehmet H. Gunes, Ph.D., Committee Member

M. Sami Fadali, Ph.D., Graduate School Representative

Marsha H. Read, Ph. D., Associate Dean, Graduate School

August, 2011

Probabilistic Trans-Algorithmic Search

Bilal Gonen

University of Nevada, Reno, 2011

Advisor: Murat Yuksel

Abstract

Online configuration of large-scale systems such as networks requires parameter optimization within a limited amount of time. This time limit is even more pressing when configuration is needed as a recovery response to a failure in the system. To quickly configure such systems in an online manner, we propose a Probabilistic Trans-Algorithmic Search (PTAS) framework which leverages multiple optimization search algorithms in an iterative manner. PTAS applies a search algorithm to determine how to best distribute available experiment budget among multiple optimization search algorithms. It allocates an experiment budget to each available search algorithm and observes its performance on the system-at-hand. PTAS then reallocates the experiment budget for the next round proportional to each algorithm’s performance relative to the rest of the algorithms. This “roulette wheel” principle favors the more successful algorithm in the next round. Following each round, the PTAS framework “transfers” the best result(s) among the individual algorithms, making our framework a “trans-algorithmic” one. PTAS thus aims to systematize how to “search for the best search”. We show the performance of PTAS on well-known benchmark objective

functions including scenarios where the objective function changes in the middle of the optimization process. To illustrate applicability of our framework to automated network management, we apply PTAS on the problem of optimizing link weights of an intra-domain routing protocol on three different topologies obtained from the Rocket-fuel dataset. We also apply PTAS on the problem of optimizing aggregate throughput of a wireless ad hoc network by tuning data rates of traffic sources. We compared the experimental results of PTAS against other three well-known search algorithms, i.e., Random Recursive Search, Simulated Annealing, and Genetic Algorithm. We observe that PTAS outperforms other three search algorithms in the majority of experiments. Especially, when the network system at-hand is very dynamic with factors, such as link failures and link recovery, PTAS performs better, and adapts to the new system more quickly due to its hybrid nature.

Acknowledgments

I would like to thank my advisor, Dr. Murat Yuksel, for his support and patience during my Ph.D. program. He spent a lot of time for me. I will never forget his generosity.

I would like to thank my Ph.D. committee members Dr. Bobby Bryant, Dr. Sami Fadali, Dr. Mehmet Gunes, and Dr. Sushil Louis for their suggestions and the time they spent for me.

I also would like to thank all of my colleagues in Computer Networking Lab including Hasan Tarik Karaoglu, Mehmet Bilgi, Mustafa Omer Kilavuz, Hakan Kardes, Suat Mercan and Engin Arslan for their help for running my experiments.

And finally, I would like to thank my wife Ayşe for her support especially in very stressful times I had during my Ph.D. studies.

BILAL GONEN

University of Nevada, Reno

August 2011

Contents

Abstract	i
Acknowledgments	iii
List of Figures	vi
Chapter 1 Introduction	1
1.1 Contributions	5
1.2 Dissertation Organization	6
Chapter 2 Related Work	8
2.1 System Modeling and Optimization	8
2.2 Optimization by Setting Parameters	13
2.3 Network Management and Configuration	14
2.4 Optimization of Ad Hoc Wireless Networks	19
Chapter 3 PTAS: Probabilistic Trans-Algorithmic Search	21
3.1 Budget Allocator with Roulette Wheel	26
3.2 Transfer of Best-So-Far Among Algorithms	29
3.3 Performance Evaluation on Benchmark Objective Functions	30

	v
3.3.1 Benchmark Objective Functions	31
3.3.2 When System Response is Fixed	36
3.3.3 When System Response Varies	44
Chapter 4 Online Optimization of Networks with PTAS	56
4.1 Experiment Setup	59
4.2 PTAS with Separate System Model	63
4.2.1 IGP Link Weights Optimization	63
4.2.2 Optimization of Ad Hoc Networks	72
4.3 PTAS with No System Model	76
4.3.1 IGP Link Weights Optimization	78
Chapter 5 Summary and Future Work	87
5.1 Future Work	88
Bibliography	91

List of Figures

1.1	Black-box optimization framework: Content of the black-box is problem-specific whilst the optimization algorithm can be generic.	2
2.1	Traffic engineering for network load balancing	17
2.2	Oscillations in routing [27].	18
3.1	Trans-Algorithmic Search for real-time system management: Each algorithm can experiment with the black-box system by trying out different parameter vectors.	22
3.2	Shrink and Re-align Process [56].	24
3.3	Budget allocation in five rounds.	27
3.4	Square Sum Function [1]	32
3.5	Visualization of Rastrigin’s function; left: surf plot in an area from -5 to 5, right: focus around the area of the global optimum at $[0, 0]$ in an area from -1 to 1. [1]	32
3.6	Visualization of Griewangk’s function; top left: full definition area from -500 to 500, right: inner area of the function from -50 to 50, bottom left: area from -8 to 8 around the optimum at $[0, 0]$ [1]	33

3.7	Visualization of Axis parallel hyper-ellipsoid function; surf/mesh plot of the function in an area from -5 to 5. [1]	34
3.8	Visualization of Rotated hyper-ellipsoid function; surf/mesh plot of the first two variables in an area from -50 to 50. [1]	34
3.9	Visualization of Ackley's Path function; left: surf plot in an area from -30 to 30, right: focus around the area of the global optimum at [0, 0] in an area from -2 to 2. [1]	35
3.10	Comparison of PTAS with RRS, SA, and GA for SquareSum function.	36
3.11	Cumulative progresses of four search algorithms for SquareSum function.	36
3.12	Comparison of PTAS with RRS, SA, and GA for Rastrigin function.	37
3.13	Cumulative progresses of four search algorithms for Rastrigin function.	37
3.14	Comparison of PTAS with RRS, SA, and GA for Griewangk function.	38
3.15	Cumulative progresses of four search algorithms for Griewangk function.	39
3.16	Comparison of PTAS with RRS, SA, and GA for Axis parallel hyper-ellipsoid function.	39
3.17	Cumulative progresses of four search algorithms for Axis parallel hyper-ellipsoid function.	40
3.18	Comparison of PTAS with RRS, SA, and GA for Rotated hyper-ellipsoid function.	40
3.19	Cumulative progresses of four search algorithms for Rotated hyper-ellipsoid function.	41
3.20	Comparison of PTAS with RRS, SA, and GA for Ackley's Path function.	41
3.21	Cumulative progresses of four search algorithms for Ackley's Path function.	42
3.22	A Drastic Change: SquareSum \rightarrow Rastrigin \rightarrow SquareSum	45

	viii
3.23 A Drastic Change: SquareSum \rightarrow Griewangk \rightarrow SquareSum.	45
3.24 A Drastic Change: SquareSum \rightarrow Axis parallel \rightarrow SquareSum.	46
3.25 A Drastic Change: Rastrigin \rightarrow SquareSum \rightarrow Rastrigin.	47
3.26 A Drastic Change: Rastrigin \rightarrow Axis parallel \rightarrow Rastrigin.	47
3.27 A Drastic Change: Griewangk \rightarrow SquareSum \rightarrow Griewangk.	48
3.28 A Drastic Change: Griewangk's \rightarrow Rastrigin \rightarrow Griewangk's	49
3.29 A Drastic Change: Griewangk \rightarrow Axis parallel \rightarrow Griewangk.	49
3.30 A Drastic Change: Axis parallel \rightarrow Rastrigin \rightarrow Axis parallel.	50
3.31 A Drastic Change: Axis parallel \rightarrow Griewangk \rightarrow Axis parallel.	51
3.32 A Small Change: SquareSum \rightarrow SquareSum-Shifted \rightarrow SquareSum.	51
3.33 A Small Change: Rastrigin \rightarrow Rastrigin-Shifted \rightarrow Rastrigin.	52
3.34 A Small Change: Griewangk \rightarrow Griewangk-Shifted \rightarrow Griewangk.	53
3.35 A Small Change: Axis parallel \rightarrow Axis parallel-Shifted \rightarrow Axis parallel.	53
3.36 A Small Change: Rotated ellipsoid \rightarrow R. ellipsoid-Shifted \rightarrow R. ellipsoid.	54
3.37 A Small Change: Ackley's \rightarrow Ackley's-Shifted \rightarrow Ackley's.	55
4.1 Optimization using a separate model of the system	58
4.2 Optimization using real-time data from the system	58
4.3 One rack of compute nodes in the UNR Research Grid [7]	61
4.4 Rocketfuel's Exodus topology.	66
4.5 Results of experiments on Exodus topology with different experiment budgets.	66
4.6 Rocketfuel's Abovenet topology	68
4.7 Results of experiments on Abovenet topology with different experiment budgets	68

	ix
4.8 Rocketfuel’s Sprint topology.	70
4.9 Results of experiments on Sprint topology with different random seeds for each run.	71
4.10 Results of experiments on Sprint topology with different experiment budgets.	71
4.11 Results of experiments on 4 nodes and 12 UDP connections with dif- ferent budgets.	74
4.12 Results of experiments on 10 nodes and 8 UDP connections with dif- ferent runs.	75
4.13 Optimization of a real-time simulator by using RRS.	80
4.14 Optimization of a real-time simulator by using Simulated Annealing.	81
4.15 Optimization of a real-time simulator by using Genetic Algorithm. . .	81
4.16 Optimization of a real-time simulator by using PTAS.	82
4.17 Comparison of PTAS with RRS, SA, and GA over different link failure frequencies.	84
4.18 Comparison of PTAS with RRS, SA, and GA over different subsequent search phase intervals	86
4.19 Comparison of PTAS with RRS, SA, and GA for using different search phase lengths and different number of rounds for PTAS	86
5.1 AS A with connections to AS B and AS C [45].	89

List of Algorithms

1	PTAS Main Function	23
2	GA Steps	25
3	Grid script	62

Chapter 1

Introduction

Although there have been several tools and outcomes [58] from research on large-scale network management, network operators have found themselves more comfortable with trusting highly-experienced well-trained human administrators. However, the complexity of the management and configuration problem is increasing due to the increasing heterogeneity in substrate technologies as well as applications' demand for more stringent performance targets. Thus, tools to achieve automated ways of managing a running network are vitally needed. In this dissertation, we present a new black-box optimization algorithm for automated management and configuration of networks.

Global optimization is a commonly used technique for close-to-optimal configuration of systems, minimization of cost [47] or maximization of benefit [32]. One method of global optimization is known as “black-box” optimization, which considers the problem-at-hand as a black box and searches for the optimal parameter setting yielding the best (i.e., minimum or maximum) output metric value. Figure 1.1 illustrates the black-box optimization approach to problem solving. As the black-box

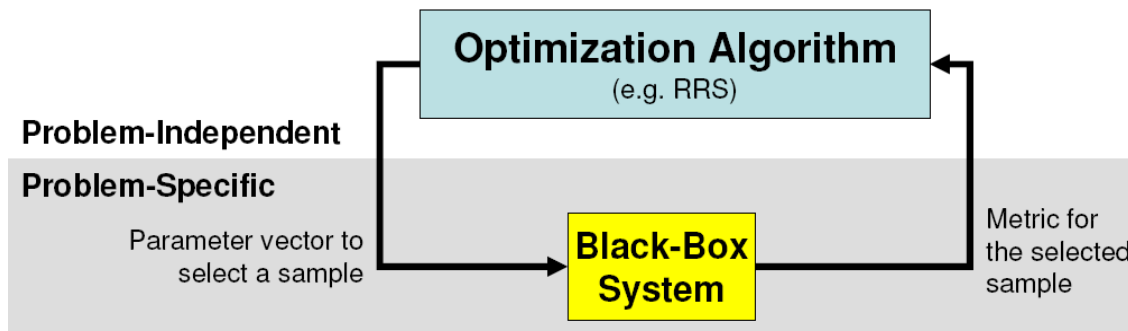


Figure 1.1: Black-box optimization framework: Content of the black-box is problem-specific whilst the optimization algorithm can be generic.

optimization is generic, the domain of applicable problems is vast as long as the output response of the problem system can be mapped to a single metric. The price of being generic and problem-independent comes typically as lack of optimality guarantees in the solution. Problem-specific methods like approximation algorithms [54], dynamic programming [48] or divide-and-conquer [12] are more successful in providing guarantees approaching the global optimum. However, these problem-specific methods may take a long time to find a solution and typically require the whole search process to complete for a solution. Most real-time systems, such as an ISP network, require parameter optimization within a limited amount of time, e.g., in response to a network failure. Thus, the optimum solution may not be reachable for the real system-at-hand within the limited time budget.

Black-box optimization algorithms must search for the optimal solution by exploring and then exploiting the response surface of the system. Exploration searches for new uncertain outcome taking risk. Exploitation refines of current outcome that is already known. Exploitation is usually a short-time process with immediate, certain benefits [6]. A critical design issue is to balance the amount of time spent on exploration and exploitation. Various black-box optimization search algorithms have been

designed with different balancing of this tradeoff, e.g., Hill Climbing [13], Simulated Annealing [28], Genetic Algorithms [18], or Random Search [61]. Some of the search algorithms may perform better than others on the same black-box problem depending on the response surface.

Automating a network’s management and configuration can be quite difficult since reconfiguration of the entire network may be needed upon a major failure. Two critical aspects of the problem are: (i) network failures must be handled very quickly and (ii) the network’s optimal configuration can be quite different than the prior ones as the failures can cause a significant change in the network’s behavior (i.e., a major change in system response). To achieve fast and adaptive network system configuration, we propose a Probabilistic Trans-Algorithmic Search (PTAS) framework which leverages multiple search algorithms in an iterative manner. PTAS allocates experiment budget (i.e., the number of experiments an algorithm can make to optimize the black-box system) to each available search algorithm and observes the success in resolving the problem. Depending on their successes, PTAS reallocates the experiment budget for the next round. We make this reallocation based on a roulette wheel approach [24] favoring the more successful algorithm in the next round. Following each round, the PTAS framework allows “transfer” of best results among the algorithms being used, which makes it a “trans-algorithmic” one.

The PTAS framework can automatically determine the best set of algorithms for the problem-at-hand. It is also adaptive to changes in the system behavior. This feature is especially useful for systems involving unexpected failures causing the response behavior to change, e.g., router or link failures in a network. We implement our hybrid search framework, by using three search algorithms, i.e., Recursive Random Search (RRS) [56], Simulated Annealing [28] (SA), and Genetic Algorithm [18]

(GA). We show that PTAS can outperform any of the search algorithms on well-known benchmark objective functions. We also experiment with changing the objective function in the middle of the optimization process and show that PTAS outperforms a singleton algorithm more pronouncedly.

We apply PTAS on the problem of interior gateway protocol (IGP) link weights optimization on a realistic ISP topology. The framework includes two cycles: optimization cycle and deployment cycle. The deployment cycle takes place at a time scale where it is possible to configure a new set of parameters in the real network system. The optimization cycle iterates through a model of the system, for which we used a network simulator, and attempts to find the best possible set of parameters to configure during the next occurrence of the deployment cycle. In that sense, the frequency of the deployment cycle is smaller than the optimization cycle. The underlying assumption in this approach is that the system-at-hand does not significantly change during at least one deployment cycle. This assumption is generally true for networks with many fixed components such as backbone networks or wireless mesh networks. We show that PTAS successfully finds intra-domain routing configuration providing better throughput.

In order to compare our PTAS algorithm with other three individual search algorithms, we run experiments on NS-2 network simulator [2] to optimize wireless ad hoc network by tuning data rates of traffic sources. The results are also promising. Because of its hybrid nature, PTAS found either the best or the second best results in most of the cases.

1.1 Contributions

In this dissertation, our contributions can be classified as follows;

- There are several algorithms tackling black-box problems, and some of these algorithms are hybrid algorithms comprised of multiple algorithms. To the best of our knowledge, our work is the only hybrid algorithm tackling black-box problem that includes an adaptive budget allocator system. Our PTAS framework presents a new approach in using a search algorithm to balance the experiment budget among multiple algorithms. We show how to hybridize three search algorithms with different characteristics such that some are good in exploration (Recursive Random Search [56] and Genetic Algorithm [18]) and others are good in exploitation (Simulated Annealing [28]).
- We apply PTAS and three other search algorithms on six well-known objective functions, and run a number of experiments by using different prime numbers as the seed for random number generators. PTAS outperforms the other three algorithms on average.
- We apply a version of PTAS with a separate model to a network problem on a realistic ISP topology, and show that PTAS not only increases the throughput considerably, but also outperforms other three search algorithms in most of the cases.
- We compare PTAS with the three search algorithms when the network system at-hand is very dynamic with factors, such as link failures and link recovery. PTAS performs better, and adapts to the new system more quickly due to its hybrid nature.

1.2 Dissertation Organization

This dissertation is organized as follows: Chapter 2 reviews relevant work in the literature. It covers some major works on optimization and system modeling. It covers relevant papers on optimization by tuning the parameters of a running system. Since we compared the performance of PTAS with other algorithms on network management problems, we also cover some relevant work on network management and configuration. Finally, we survey papers on the optimization and management of ad hoc wireless networks.

Chapter 3 details the PTAS framework. It compares the performance of the PTAS with three other algorithms, namely Recursive Random Search [56], Simulated Annealing [28], and Genetic Algorithm [18], on some benchmark objective functions. To compare the performance of PTAS with the three algorithms when the system is dynamic, we use a number of the objective functions as benchmarks. To mimic small changes in the system, sometime during the search, we shift the objective function by adding some constant value so that the objective function's response surface is moved with a constant distance. To mimic drastic changes in the system, i.e., some core links fail, we substantially change the objective function while the optimization search is going on. We show how adaptive the PTAS is against such small or drastic changes.

Chapter 4 discusses the experimental results of PTAS. In this chapter, we apply PTAS for online optimization of a running network. To perform the online network optimization, we follow two different approaches: *Separate System Model* and *No System Model*. We explain the two approaches, and then we explain the experimental environment and details of the network simulator in Section 4.1. In Section 4.2,

we discuss the experiment results of PTAS with separate system model on two well-known network problems; (i) problem of interior gateway protocol (IGP) link weights optimization on realistic ISP topologies (ii) optimization of ad hoc wireless networks by tuning the data rates of traffic sources. Section 4.3 discusses the experiment results of PTAS with no system model on a real-time running interior gateway protocol (IGP) link weights optimization problem. To make the network system sufficiently dynamic, we fail and recover some links in the network, and we evaluate and compare the results of experiments.

Finally, we summarize our work and give our ideas about future directions in Chapter 5.

Chapter 2

Related Work

This chapter reviews the literature related to our work. In this chapter, we categorize the related work into four main groups:

- System Modeling and Optimization
- Optimization by Setting Parameters
- Network Management and Configuration
- Optimization of Ad Hoc Wireless Networks

2.1 System Modeling and Optimization

System modeling and optimization have been very fruitful fields of research resulting in significant advances in science and engineering. Numerous methods have been developed to tackle hard optimization problems in practical systems as well as to model them. Several ways of classifying optimization techniques exist, such as global versus local, discrete versus continuous, empirical versus non-empirical, partial versus

complete. In the discrete optimization, variables used in the objective function are assumed to be integer numbers, whereas in the continuous optimization, variables used in the objective function can be real numbers. Global optimization is a technique trying to find the global optimum sample in the search space. Local optimization tries to find a local optimum, and there is no guarantee to get the global optimum by using this technique. Finding the global optimum is more challenging than finding a local optimum sample.

Empirical optimization technique is based on empirical search on the objective function. This empirical search depends on trials and errors experience. Non-empirical optimization is based on deductive reasoning. Deductive reasoning is a method to reach a conclusion from a set of premises. A common example; “All men are mortal. Socrates is a man. Therefore, Socrates is mortal”.

From a systems management perspective, optimization means to determine the area of the parameter space which gives the “best” performance response. In terms of being complete or partial solutions, techniques like Exhaustive Search, Local Search [41], and Linear Programming [15] belong to the former group, and give useful, though sub-optimal results, even though the solution process is interrupted. There are several well-known techniques belonging to the latter group, such as greedy algorithms, divide-and-conquer strategies, Dynamic Programming [10] and A* algorithms [37].

Divide-and-conquer strategies are very commonly used in the cases where splitting and merging the main problem is less costly than solving the main problem itself. Dynamic programming is a more enhanced version of greedy techniques in that it employs a forward-and-backward evaluation of the candidate solutions instead of just their forward evaluation. Dynamic Programming techniques guarantee a global so-

lution while greedy techniques cannot. However, greedy techniques are more generic while Dynamic Programming is more problem-specific, because it requires the creation of solution stages.

Greedy techniques fail to guarantee globally optimum solution. The main reason is the fact that a series of greedy selections step may simply not be the overall optimum. To avoid such cases, A* algorithms employ an evaluation function (for the greedy step) that gives more information about the real value of the possible selections. Even though A* algorithms guarantee reaching the global solution, they typically require leveraging of problem-specific information in order to define the heuristic for greedy steps.

A large set of techniques for complete solutions is the class of evolutionary algorithms [9] or heuristic search algorithms, which we consider in contrast to approximation algorithms [54] for NP-hard problems. Given specific NP-Hard optimization problems, one can design approximation algorithms guaranteed to reach an approximate solution within a pre-defined error from the best solution in polynomial time. Randomized algorithms [38] probabilistically reach the pre-defined error bound in polynomial time. In contrast, heuristic search algorithms do not provide such error bounds or polynomial time bounds, but are applicable to a wide variety of problems and provide attractive performance in practice. Through this competitive selection and random variation (i.e. genetic relationship) process, the society of fellow solutions become better, and eventually include the best solution. Simulated Annealing [28], Genetic Algorithms [18] as well as several other heuristic algorithms working on complete solutions can be thought of special cases of the general class of evolutionary algorithms. Continuous optimization problems also leverage hybrid methods such as combination of gradient search methods with evolutionary techniques to achieve

faster convergence [21].

System optimization is a vast field with a significant body of prior work. There have been many studies exploring methods for system optimization, of which many have been applied to real-world problems. However, not all of these techniques are suitable for online optimization. The most common design approach of these techniques is to be *greedy and exploitive*, e.g., steepest descent or Newton’s gradient methods [33]. Most real-world applications of online optimization have dealt with software or web systems with a *passive and sample-based* approach [20] to model the recent behavior of the system at hand.

Boutilier et al. [11] present a new framework for a banner advertising auction. They describe a general model of expressive ad contracts and rerun the channel allocation mechanism periodically. Similarly to our work, they also use an allocation model which is executed in real time.

Diao et al. [14] introduced a framework including several algorithms. They apply it to a running system to reconfigure it by setting the parameters. They also apply their framework to black-box problems without prior knowledge about the system being optimized. In this paper, the authors show that they can optimize a running database system for e-commerce applications with respect to the system’s response time. In common with our work, they also propose a generic approach to automating the optimizing configuration parameters task. By generic, we mean that the approach is relatively independent of the target system for which the optimization is done. Unlike our approach, they handle interdependencies between the configuration parameters. In our work, however, we assume that the system is completely black-box, and there is no prior knowledge of whether there is any interdependency between the configuration parameters.

In their paper [34], Liu et al. introduce a method to maximize profits in an e-commerce environment. When creating their model, the output metric they use is revenue. This revenue is based on the Quality-of-Service (QoS) criterias of Service-Level Agreements. When creating the cost model, they check to see if Quality-of-Service (QoS) guarantees are satisfied or not. This paper also differs from our approach since they require a detailed knowledge of the system being optimized.

In their work, Menasce et al. [36] describe a system that performs on-line optimization of a web server by using hill climbing techniques. We did not use hill climbing in our PTAS framework, but we used a simulated annealing search algorithm, which is similar to hill climbing algorithm. Another similarity is that they also search for the best combination of configuration parameters as we do. Their approach is different from our approach because their approach requires a detailed knowledge of the system being optimized in order to construct the queuing models.

Similarly, Rao et al. [44] and Lohman et al. [35] applied apriori system knowledge to automate configuration of databases in an online manner. These approaches are essentially different than ours as they require some prior knowledge of the target system being optimized and follow a white-box approach, whereas our two-phase approach requires no prior knowledge of the system and strictly follows a black-box approach to allow solving general problems.

Tronci et al. [52] eliminate some of the experts first to determine which experts to pick. In contrast, in our work, we always use three algorithms for hybridization, with varying percentile in each round. In their work, they present an experimental analysis to evaluate the correlation between the combination methods and performance measures. They also show the behavior of the combination methods in terms of different measures.

2.2 Optimization by Setting Parameters

In many cases, complex real systems have large parameter ranges which generally causes combinatorial explosion in the number of their possible metric outcomes. The challenge is to search through such large parameter spaces with minimal trials to get maximal information about the global solutions. Several realistic systems, such as network protocols in the Internet, can easily have parameter counts in the order of millions. The design challenge has been to find the right balance between exploitation and exploration of parameter spaces. In this respect, hybrid designs have received considerable attention from researchers. Such hybrid designs included merging of a genetic algorithm (GA [18]) with a local search strategy based on the interior point method [25], using GA [18] for global exploration and Ant Colony Optimization (ACO) for local exploitation [29], combining a calculus-based method with GA [22], mixing GA with Tabu Search [42] for solving resource scheduling problems [60], mixing ACO with Simulated Annealing [28] for cluster analysis [40]. In [51], the authors develop a hybrid method combining Adaptive Partition-based Search (APS) [26] and Downhill Simplex Algorithm (DSA) [39], where APS is used for exploration and DSA for exploitation. Another hybridization was proposed in [59], between GA and a stochastic variant of the simplex method [39].

A crucial component of optimizing a running system is the set of algorithms being used for system optimization. Evolutionary algorithms [9] or heuristic search algorithms are typical approaches to optimizing a large-scale system with many parameters such as networks. Our PTAS framework presents a new approach in using a search algorithm to find a good way of balancing the experiment budget among multiple algorithms.

In the second version of PTAS, which is PTAS with no system model, we have optimization cycle and deployment cycle. In most systems, deployment of new configurations to the system may cause severe problems, failures, or a sudden drop in the performance. Raghavachari et al. [43] covers two case studies for configuration of parameter settings for application servers. The case studies are “manufacturing application” and “Trade, a brokerage application”. First, they apply exploration technique to the search space simply randomly generating numbers for parameters. Then they analyze the results to determine which parameters are the most relevant and further explore those parameters.

Hutter et al. [23] describe an automatic framework for the algorithm configuration problem. They provide methods for optimizing a target algorithms performance on a given class of problem instances by varying a set of parameters. They cover a group of local-search-based algorithms’ configuration procedures and introduce some techniques to optimize them by adaptively changing the time to spend for individual configurations. They studied the problem of automatically configuring the parameters of complex, heuristic algorithms in order to optimize performance on a given set of benchmark instances. Their idea is similar to our budget allocator part of the PTAS, however the calculations are different.

2.3 Network Management and Configuration

The internet consists of many routing domains composed of one or more autonomous systems (ASes). An AS may have hundreds or thousands of links and routers. Autonomous systems are configured by an internet service provider or a company which continuously performs traffic engineering within its domain to increase the efficiency

of its networks.

The goal of intra-domain traffic engineering is to utilize the network resources in an autonomous system (AS) more efficiently. Interior Gateway Protocols are used to calculate the shortest paths in an autonomous system where the traffic flows are directed. There are two common Interior Gateway Protocols: OSPF (Open Shortest Path First) and IS-IS (Intermediate System-Intermediate System). Interior Gateway Protocols (IGPs) direct traffic based on link weights assigned by the network operator. Routers in an autonomous system calculate the shortest paths to others routers in the autonomous system, then add these values in their destination tables. By tuning the IGP link weights, the network operators try to change what these routers calculate as the shortest paths and balance the traffic load on individual links of their ASes. This is a quite complicated problem. For example, the OSPF weight-setting problem is shown to be NP-hard [16].

Here is how OSPF calculates routes. Integer numbers between 1 and 65535 are assigned to all of the links in a network as the link weights. The weight of a path is the sum of the link weights of the path. According to the OSPF algorithm, each router computes a tree of shortest paths rooted at the router. By using this tree-based graph, routers can retrieve the cheapest paths to all other routers in the autonomous system. OSPF requires routers to exchange routing information with all the other routers in the AS.

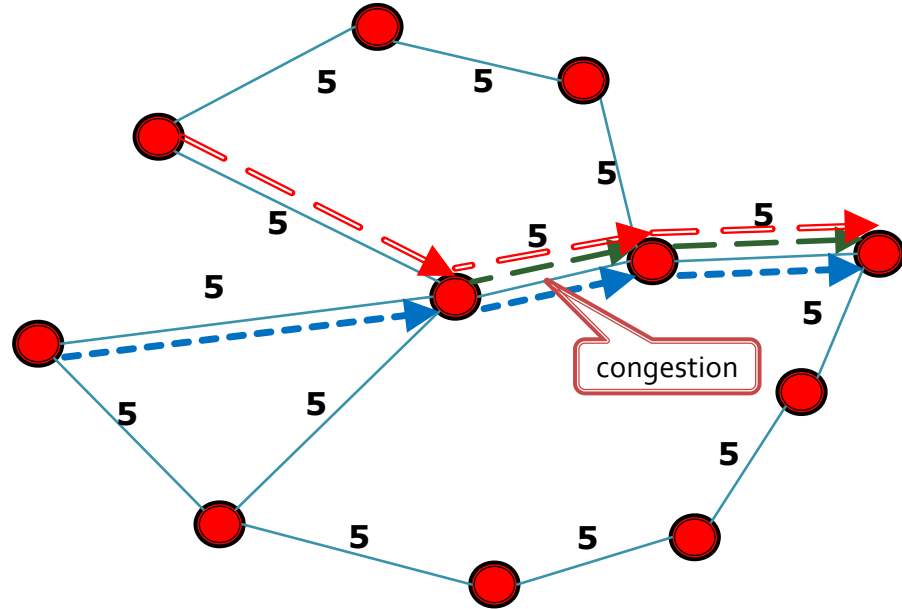
Figure 2.1-a presents an example scenario where three traffic flows are routed over a common path, causing overloading. Assume that all three traffic flows have a transmission data rate 10 Mbps. Also assume that the middle link labeled “congestion” has a bandwidth of 15 Mbps. When all the links are assigned a link weight of 5, then all three traffic flows will pass through the link in the middle according

to shortest path calculation. The total traffic flow of 30 Mbps cannot pass through the link with 15 Mbps bandwidth. Thus, link congestion occurs and packets drop. Figure 2.1-b shows how intra-domain traffic engineering helps route the traffic to different paths and balances the network load.

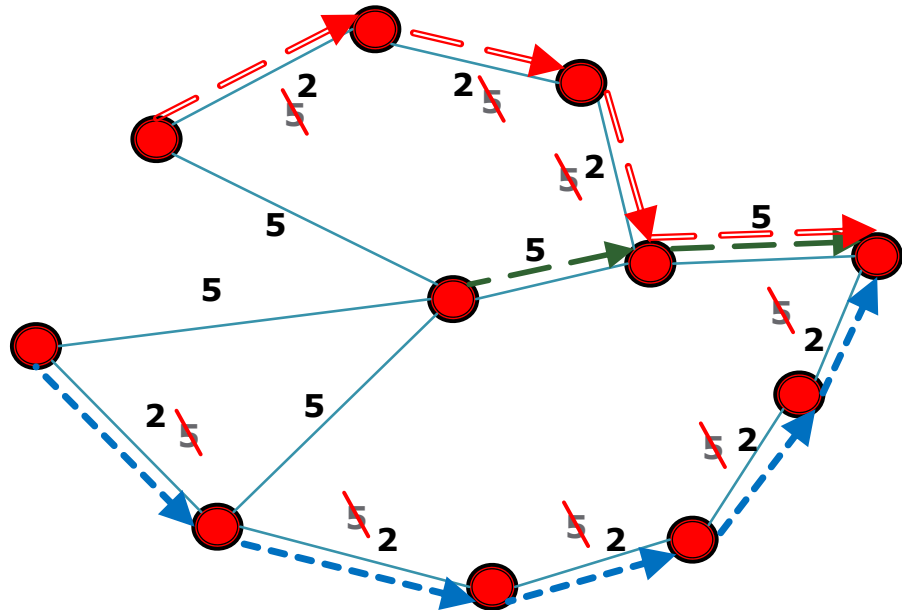
A question may arise as; “Why not simply increase the cost of the congested link making the cost proportional to the traffic flow?”. As illustrated in Figure 2.2, doing so may cause oscillations in the system, and damages the hardware.

Automated network management and configuration has been of high interest in the research and ISP communities [58]. In [16], the authors proposed a solution to set the link weights of interior gateway protocols (IGPs) according to the given network topology and traffic demand so as to control intra-domain traffic and meet traffic engineering objectives. The IGP link weight setting problem is known to be NP-hard [16]. In [46], Riedl et al. adopted objective of routing optimization as the maximum link utilization in the network. The latest development on the IGP link weight setting problem is RRS [58], and we show that PTAS outperforms RRS on this critical problem.

Wang et al. [55] proposed an algorithm to tune IGP link weights. By tuning the IGP link weights, they optimize a multimedia IP network. To better utilize the network, internet service providers use multicast for videos to their clients. They use the same network to use unicast for their clients. Their service traffic is sent from source to receivers. The IGP shortest path calculation is done from source to receivers for unicast traffic but from receivers to source for multicast traffic. To minimize congestion, they tune IGP link weights such that the traffic flows for multicast and unicast do not overlap. Their approach enables service providers to use additional capacity on the reverse direction of multicast traffic flow.



(a) Congestion occurs because three traffic flows are routed over a common path.



(b) Traffic engineering ensures the network load balancing by directing the traffic flows onto different paths.

Figure 2.1: Traffic engineering for network load balancing

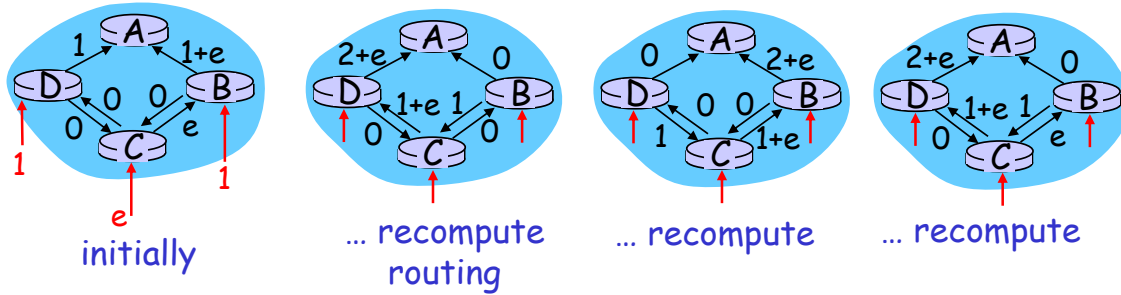


Figure 2.2: Oscillations in routing [27].

Wang et al.'s algorithm is not a stochastic search algorithm. Given at least two connected network topologies, their algorithm sets link weights in an iterative way. They start with an empty graph G , select a subset of links from the network, set link weights for the selected links, and add them to the graph G . They assign either high or low values to the weights. The low weight could be one or other numbers. The high weight should be relatively large, for example, the number of the links in the network times the low weight. This procedure keeps going until all of the links of the network are added to the graph. In a multimedia IP network, they assume the multicast routing is using reverse path forwarding technique, and the IGP unicast routing protocol is using shortest path routing [55].

Sridharan et al. [49] developed a method for link failures. They compute an optimum set of links weights which will perform well for all single link failures in an interior gateway protocol. They observe that link failures are usually short-lived, and quickly recovered, but may occur frequently. Hence, for a large network topology, instead of spending time searching for an optimum set of link weights and affecting the network's performance during the optimization phase, it is more reasonable to compute the link weights before failure, and deploy those link weights as soon as the failure occurs.

Ye et al. [57] address the issues associated with the dynamic optimization of link weights setting. In our work, we maximize aggregate network throughput. However, they chose the packet loss rate in the network as the optimization metric. The packet loss rate was formulated in terms of the link parameters, such as bandwidth and buffer space, and the parameters of the traffic demands. They used a queuing model to compute the packet drop probability on a given link in the network. Similar to our work, they also optimize the network by tuning the OSPF weights but they do not use multiple algorithms to hybridize.

2.4 Optimization of Ad Hoc Wireless Networks

In this section, we cover some literature related to optimization of ad hoc wireless networks. As opposed to the wired networks, ad hoc wireless networks do not have preexisting infrastructure. They don't have routers. Nodes participate in routing by forwarding the packets to their neighbor nodes. The distance that nodes can send their packets depends on the transmission power level of the nodes. Transmission power level and carrier sense threshold have a direct effect on the overall network capacity. When the transmission power level is low, then less data is received at the destination. If transmission power level is too high, then it causes interference. Also increasing carrier sense threshold reduces the carrier sense range. In their paper, Kim et al. [50] study the problem of ad hoc network optimization by finding some optimum values for the transmission power level and the carrier sense threshold. We applied PTAS to ad hoc network optimization by tuning the data rates of traffic sources, but as a future work, we are also planning to tune transmission power level and carrier sense threshold for optimization.

Liu et al. [31] optimize the energy efficiency of wireless ad hoc networks by tuning the transmission power. They present two algorithms for neighbor selection. There are numerous topology control protocols with the goal of increasing the throughput of the wireless ad hoc networks and while reducing its energy consumption. Most of them use some type of pruning algorithm to remove neighbors from their neighbor list while still maintaining network connectivity. In their paper, they claim as the main contribution of the paper that the algorithms consider energy consumption of the irrelevant receivers as well as the energy consumption at transmitter and receiver at the destination.

In [8], Aron et al. introduce a distributed topology control algorithm to reduce the energy consumption in wireless mesh networks. Similar to our work, they also use NS-2 network simulator to assess their algorithm. Unlike ours, their algorithm is not a stochastic search algorithm. When they calculate the transmission power for each node, also in order to not to lose network connectivity, they use their local information about the neighborhood. That is, the optimization problem is not a black-box problem for their algorithm.

Yi Li et al. [30] developed an algorithm that calculates an upper limit for transmission data rates of traffic sources for wireless network optimization. As the input for the algorithm, they use traffic flow demands. As for the performance measures, they consider aggregate network throughput and also maximizing fairness.

Chapter 3

PTAS: Probabilistic Trans-Algorithmic Search

The essence of PTAS is to apply an optimization search algorithm to find out how to best distribute available experiment “budget” to multiple optimization search algorithms. In other words, PTAS aims to systematize how to “search for the best search”. First, we split our total budget into smaller chunks, and call them “round budgets”. In the first round, the budget allocator in the PTAS framework (Figure 3.1) allocates the round budget among the three algorithms equally. At the end of the first round, it compares the responses returned from each algorithm. When it allocates the round budget for the second round, it allocates the round budget among the three algorithms based on how they performed in the first round. When allocating the round budget for the third round, it allocates the round budget among the three algorithms based how they performed in the second round, and so on. At the end of each round, we transfer/exchange the best result(s) among the algorithms.

Total Budget: This is the total budget that we use for our experiment. To

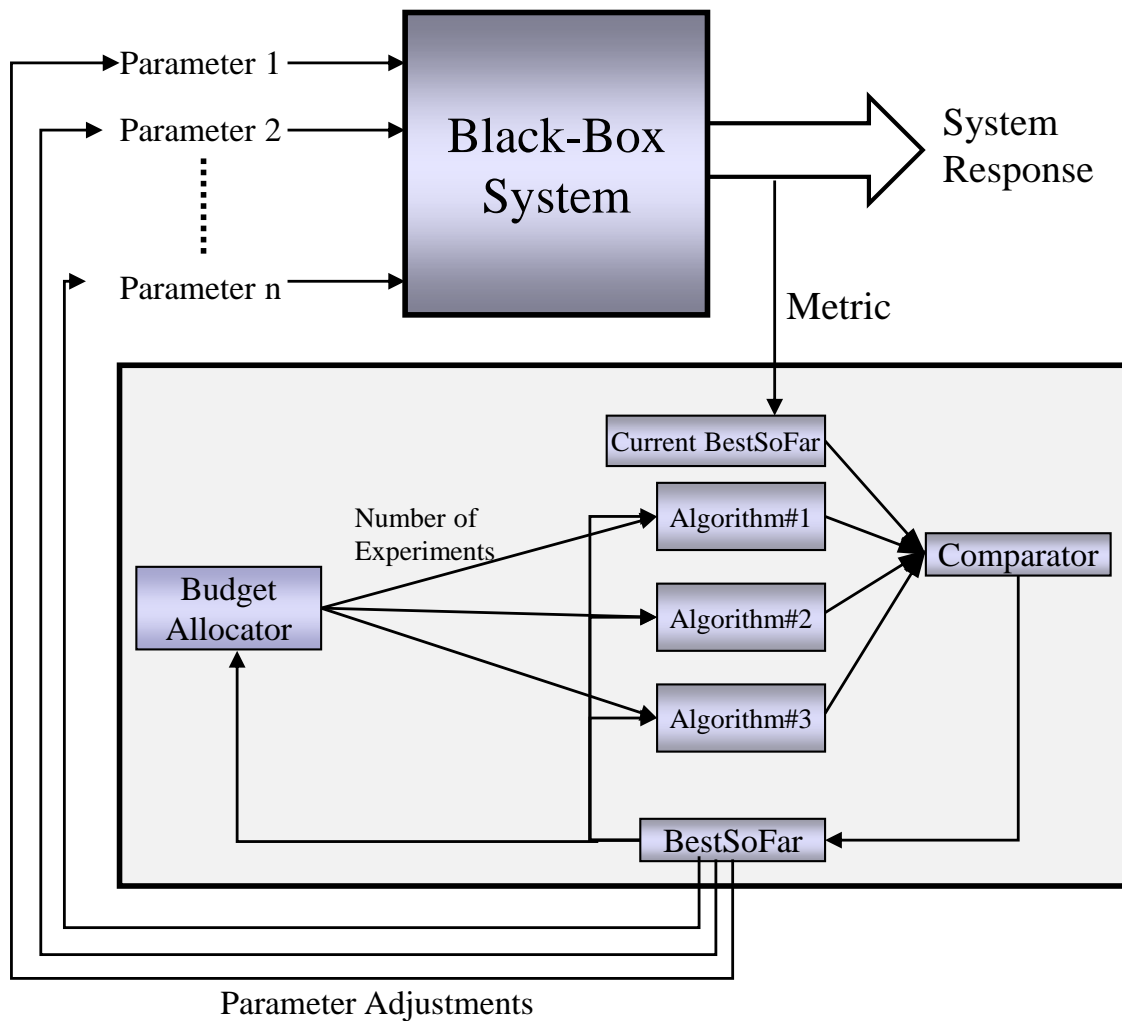


Figure 3.1: Trans-Algorithmic Search for real-time system management: Each algorithm can experiment with the black-box system by trying out different parameter vectors.

Algorithm 1 PTAS Main Function

```

1: while  $budget > 0$  do
2:    $best1 \leftarrow Algorithm1(roundbudget1)$ 
3:    $best2 \leftarrow Algorithm2(roundbudget2)$ 
4:    $best3 \leftarrow Algorithm3(roundbudget3)$ 
5:   recalculate round budgets based on performances
6:   transfer  $bestOf(best1, best2, best3)$  to all three algorithms
7: end while

```

calculate the output of an objective function for some given parameters, we use *eval* function. Calling *eval* function one time costs one unit budget.

Budget for a round: This is the budget that we use for each round in our program. Initially we split this amount equally between three algorithms (RRS [56], SA [28], and GA [18]). This budget does not change for each round; however the budget each algorithm can get in a given round is based on how they performed compared to previous rounds.

For our PTAS experiments in this dissertation, we used three algorithms as the individual algorithms available within the PTAS framework:

Recursive Random Search (RRS) [56]: There are two elements in a stochastic search algorithm, i.e., *exploration* and *exploitation*. Exploration examines the macroscopic features of the search space and tries to determine promising areas in the parameter space. Exploitation focuses on the microscopic features of the search space and tries to improve the solution in that local search space.

The RRS algorithm uses random sampling for exploration and recursive random sampling for exploitation. In the exploration phase, it spends some of its budget to identify a promising area. After the exploration phase, it goes into exploitation phase around the promising area found in the exploration phase. In the exploitation phase, there are two methods, i.e., *re-align sub-phase* and *shrink sub-phase*. Random

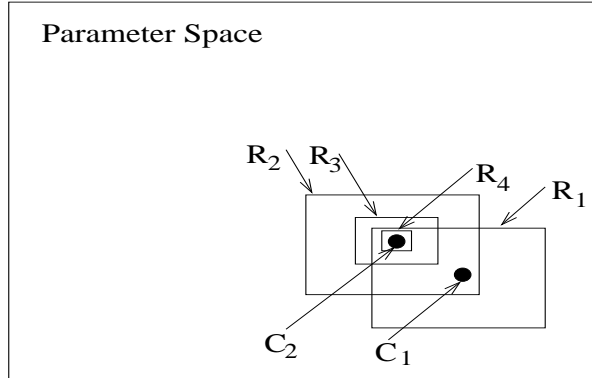


Figure 3.2: Shrink and Re-align Process [56].

sampling is also used in the exploitation phase recursively. As depicted in figure 3.2, the sample space is re-aligned or shrunk until its size falls below a predefined level.

In figure 3.2, 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 . Then the sample space is moved from R_1 to the neighborhood R_2 of C_2 .

If random sampling fails to find a better point in some predefined number of samples, RRS reduce the size of the search space in exploitation phase, which is called “shrink operation”. With re-align and shrink alternately performed, the sample space converges to the local optimum eventually. For example, in figure 3.2, after some predefined number of unsuccessful samples in R_2 , the sample space is shrunk to R_3 , then to R_4 if sampling in R_3 continues to fail. The whole exploitation process continues until the size of sample space falls below a certain threshold [56].

Simulated Annealing (SA) [28]: The Simulated Annealing algorithm is inspired by physical annealing in metallurgy. Annealing is a process used in metallurgy to reduce the defects of a material by heating it first and then cooling it [4]. In the Simulated Annealing algorithm, an initial temperature value is set. With some

probability dependent on the temperature, Simulated Annealing algorithm accepts some point even if the value of the point is worse than the current location. This probability is higher as the temperature is high. As the search progresses, the temperature is cooled down at every iteration. When the temperature is very high, the SA algorithm behaves like a random walk algorithm. As the temperature goes to zero, the algorithm behaves like a hill climbing algorithm.

Genetic Algorithm (GA) [18]: Genetic algorithm is a search algorithm which is inspired by evolutionary biology. A population in a Genetic Algorithm consists of individuals which represents a candidate solution for the problem. The steps of a GA are shown in Algorithm 2.

Algorithm 2 GA Steps

- 1: Choose initial population
 - 2: Evaluate the fitness of each individual in the population
 - 3: **while** <terminating condition> **do**
 - 4: Select best-ranking individuals to reproduce
 - 5: Breed new generation through crossover and mutation (genetic operations) and give birth to offspring
 - 6: Evaluate the individual fitnesses of the offspring
 - 7: Replace worst ranked part of population with offspring
 - 8: **end while**
-

In the initialization phase of the GA, a predefined number of individuals are created. An individual has some number of chromosomes. In our implementation of GA, each chromosome corresponds to a parameter. In the initialization phase, we randomly pick an integer number between the lower and upper limits for the parameters, and then assign it to chromosome of the individual. We used 20 as the population size. We assume that a round budget is much larger than the population size. In the selection phase of our GA, we select the two best ranked individuals from the population as parents. For the crossover operation in our GA implementation,

we randomly pick a crossover point, and then apply the crossover operation on the integer parameters of the individuals. At the end of this crossover operation, two offsprings are bred. Then with some probability, we mutate the new offsprings by changing one of their chromosomes. We select the two best ranked individuals among the two parents and two offsprings. Then, we replace worst ranked two individuals in the population with the two best ranked individuals obtained from the crossover operation.

3.1 Budget Allocator with Roulette Wheel

The crucial component of PTAS is to intelligently allocate the budget to individual algorithms so that the overall search performance is satisfactory. We use a “roulette wheel” [24] method to re-allocate the round budget among the individual algorithms. We call the PTAS component doing this budget distribution as “budget allocator” (see Figure 3.1).

The budget allocator first splits the round budget into two portions: *hard* (RB_{hard}) and *soft* (RB_{soft}). The hard round budget is the portion that has to be equally distributed among the individual algorithms, and the soft round budget is the portion that can be unequally distributed. The intuition behind this design is to give each algorithm a chance to perform in the next round. In her article [53], Pamela J. Vaccaro states that “*Simply put, the 80/20 rule states that the relationship between input and output is rarely, if ever, balanced. When applied to work, it means that approximately 20 percent of your efforts produce 80 percent of the results. Learning to recognize and then focus on that 20 percent is the key to making the most effective use of your time.*”. Therefore, in this dissertation, we picked a soft round budget

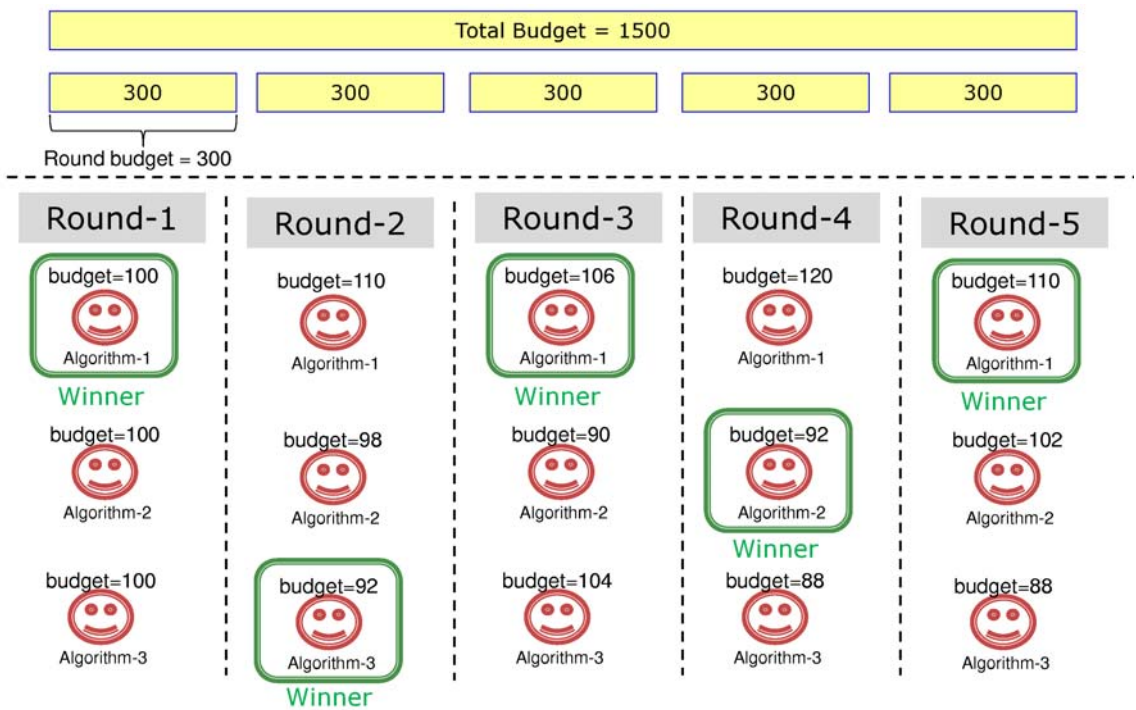


Figure 3.3: Budget allocation in five rounds.

portion of 20%, which is a relatively conservative apportionment since only 20% of the whole round budget is reallocated each time. More aggressive budget allocation can be applied by increasing the soft round budget portion. Investigation of this matter is for future work.

The soft round budget is where the roulette wheel idea is applied. We distribute the soft round budget among the individual algorithms considering the percentage improvement they achieved in the previous round. Figure 3.3 illustrates a sample scenario for how roulette wheel affects the budget allocation among three algorithms.

We now describe how to apply the roulette wheel technique on the soft round budget allocation. Let the round budget be $B = \beta_{RRS}[i] + \beta_{SA}[i] + \beta_{GA}[i]$, where $\beta_{RRS}[i]$, $\beta_{SA}[i]$, and $\beta_{GA}[i]$ are round i budget of RRS, SA, and GA respectively. Further, let the best-so-far value found during round i be $\Delta[i] = \min(\delta_{RRS}[i], \delta_{SA}[i], \delta_{GA}[i])$, where $\delta_{RRS}[i]$, $\delta_{SA}[i]$, and $\delta_{GA}[i]$ are the best-so-far value found during round i by RRS, SA, and GA respectively. We first calculate percentage improvement for each algorithm, as shown for RRS below:

$$T_{RRS} = \frac{\delta_{RRS}[i-1] - \delta_{RRS}[i]}{\delta_{RRS}[i-1]} \times 100 \quad (3.1)$$

$$T[i] = T_{RRS}[i] + T_{SA}[i] + T_{GA}[i] \quad (3.2)$$

The fractions are set to 1 initially. The expression below shows how to calculate fraction for RRS:

$$F_{RRS}[i] = F_{RRS}[i-1] + \frac{T_{RRS}[i]}{T[i]} \times RB_{soft} \times 3 - RB_{soft} \quad (3.3)$$

To calculate the round budgets for the next round, we multiply the current round budgets with fractions ($F_{RRS}[i]$, $F_{SA}[i]$, and $F_{GA}[i]$) for each algorithm:

$$\beta_{RRS}[i] = \beta_{RRS}[i-1] \times F_{RRS}[i] \quad (3.4)$$

For example, assume that RRS improved from $\delta_{RRS}[i-1]=1000$ to $\delta_{RRS}[i]=800$ during round i . Then, by using equation 3.1, $T_{RRS}[i]$ is $(1000-800)/1000*100 = 20$. Further, let $T_{RRS}[i]=20$, $T_{SA}[i]=10$, and $T_{GA}[i]=0$ and let the round budgets in the current round be all equal to 100, i.e., $\beta_{RRS}[i]=100$, $\beta_{SA}[i]=100$, $\beta_{GA}[i]=100$. Using equation 3.3,

$$\begin{aligned} F_{RRS}[i+1] &= 1 + (20/30) \times 0.2 \times 3 - 0.2 = 1.2 \\ &= 1 + 0.4 - 0.2 = 1.2 \end{aligned}$$

Using equation 3.4,

$$\begin{aligned} \beta_{RRS}[i+1] &= \beta_{RRS}[i] \times F_{RRS}[i+1] \\ \beta_{RRS}[i+1] &= 100 \times 1.2 = 120 \end{aligned}$$

In more simple terms, RRS gets back two third of the total soft round budgets $60*20/30 = 40$, SA gets back one third $60*10/30 = 20$. Because percentage improvement of GA is zero, GA gets back nothing $60*0/30 = 0$. After this allocation, the round budgets become 120, 100, and 80 for RRS, SA, and GA respectively.

3.2 Transfer of Best-So-Far Among Algorithms

In PTAS, we run the three algorithms (RRS [56], SA [28], and GA [18]) at each round. At the end of each round, we get best result of these three algorithms. If it is better than current best-so-far, we update the best-so-far sample (which is a parameter vector) and transfer it into these three algorithms. In other words, if one of the algorithms found a very good sample in the previous round, the other

algorithms also benefit from that sample by incorporating it into their search. How this is accomplished is different for each algorithm.

For the RRS [56], in each round, we spend some of our budget for the exploration phase. Then we begin the exploitation phase around the point which has the best value found in the exploration phase. If we cannot find any point better than the best-so-far value in the exploration phase, then we begin the exploitation phase around the best-so-far. This is how we transfer the best-so-far sample to the RRS algorithm.

For the SA [28], in the first round, we randomly pick a point as the initial state for the simulated annealing. After the first round, as the initial state of a round, we use the best result found in the previous round. This is how we transfer the best result of the three algorithms to the simulated annealing algorithm.

For the GA [18], in the first round, we generate samples (i.e., individuals) randomly to form our population. Then at the end of each round, we find the worst individual in the population, replace it with best-so-far. This is how we transfer the best result of the three algorithms to genetic algorithm.

3.3 Performance Evaluation on Benchmark Objective Functions

In order to compare our PTAS with separate system model algorithm with other three individual search algorithms, we use several benchmark objective functions in the system. These benchmark objective functions are explained in 3.3.1, and the results of experiments are discussed in 3.3.2 and 3.3.3.

3.3.1 Benchmark Objective Functions

In order to compare our PTAS algorithm with other three individual search algorithms, we used several benchmark objective functions in the system. These are; Square Sum function (Figure 3.4), Rastrigin function (Figure 3.5), Griewangk's Function (Figure 3.6), Axis parallel hyper-ellipsoid function (Figure 3.7), Rotated hyper-ellipsoid function (Figure 3.8), and Ackley's Path function (Figure 3.9). All of these six objective functions have the global optimum at zero.

$$f_1(x) = \sum_{i=1}^n x_i^2 \quad -5.12 \leq x_i \leq 5.12$$

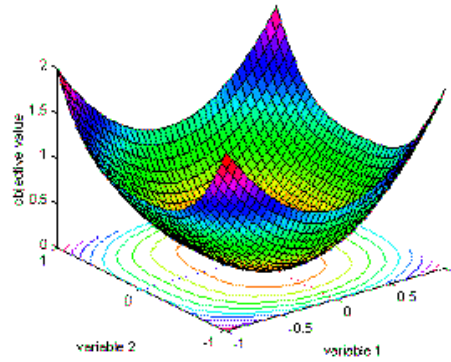
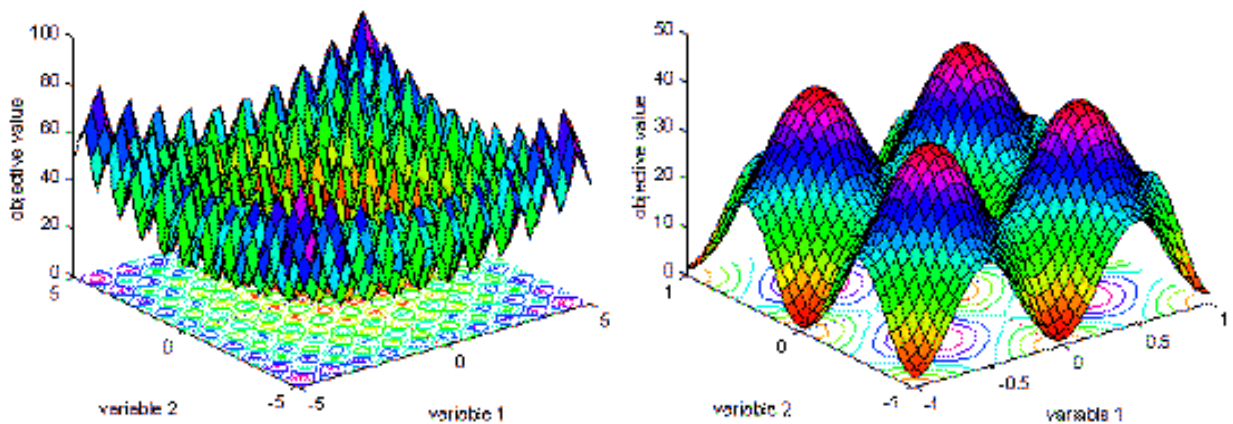


Figure 3.4: Square Sum Function [1]

$$f_6(x) = 10 \cdot n + \sum_{i=1}^n (x_i^2 - 10 \cdot \cos(2 \cdot \pi \cdot x_i)) \quad -5.12 \leq x_i \leq 5.12$$

Figure 3.5: Visualization of Rastrigin's function; left: surf plot in an area from -5 to 5, right: focus around the area of the global optimum at $[0, 0]$ in an area from -1 to 1. [1]

$$f_8(x) = \sum_{i=1}^8 \frac{x_i^2}{4000} - \prod_{i=1}^8 \cos\left(\frac{x_i}{\sqrt{i}}\right) + 1 \quad -600 \leq x_i \leq 600$$

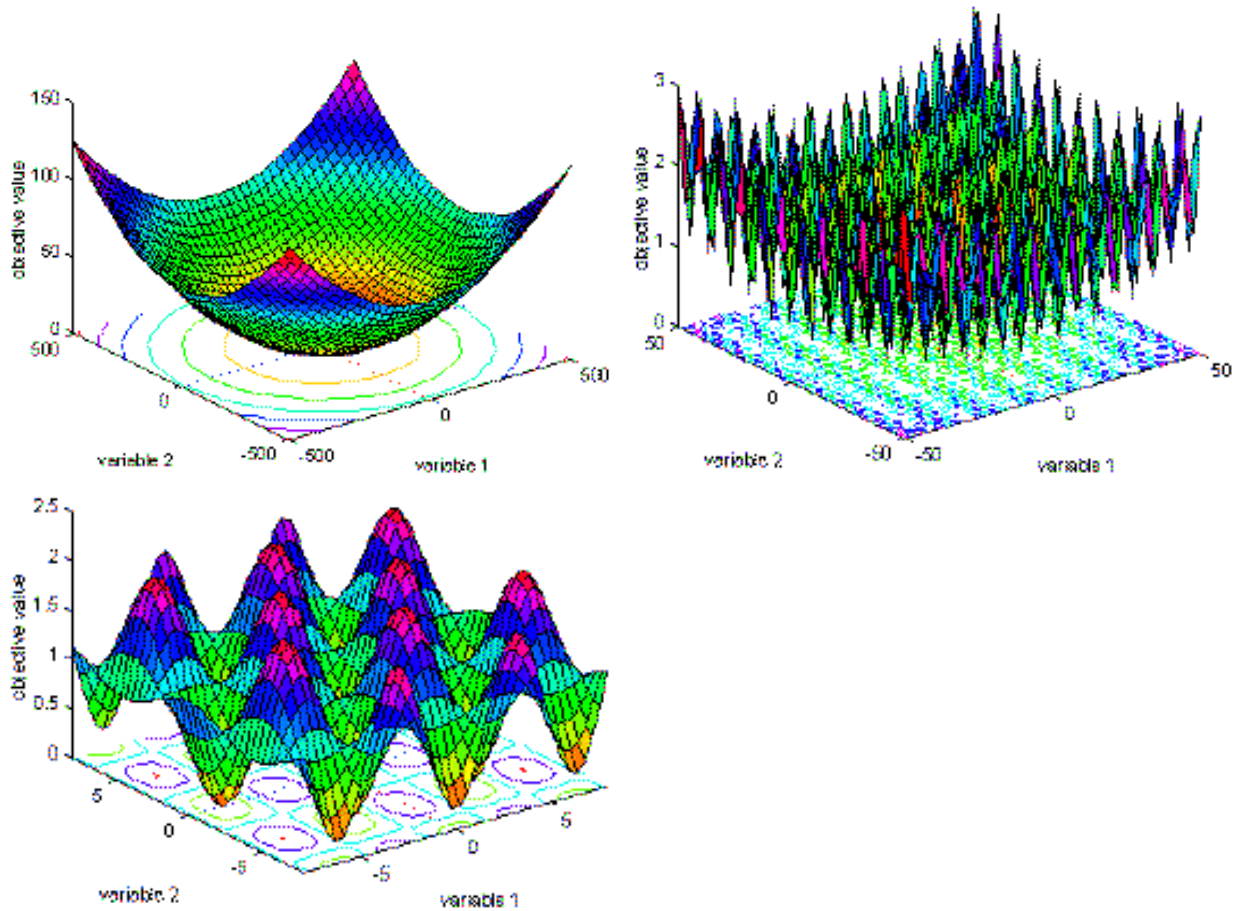


Figure 3.6: Visualization of Griewangk's function; top left: full definition area from -500 to 500, right: inner area of the function from -50 to 50, bottom left: area from -8 to 8 around the optimum at [0, 0] [1]

$$f_{1a}(x) = \sum_{i=1}^n i \cdot x_i^2 \quad -5.12 \leq x_i \leq 5.12$$

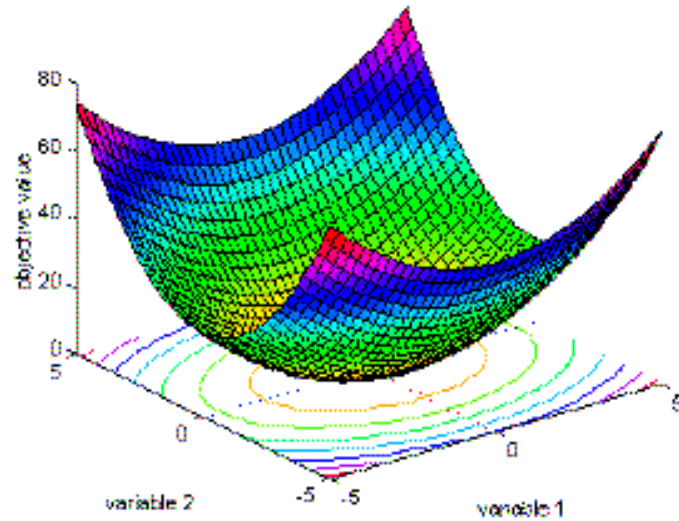


Figure 3.7: Visualization of Axis parallel hyper-ellipsoid function; surf/mesh plot of the function in an area from -5 to 5. [1]

$$f_{1b}(x) = \sum_{i=1}^n \left(\sum_{j=1}^i x_j \right)^2 \quad -65.536 \leq x_i \leq 65.536$$

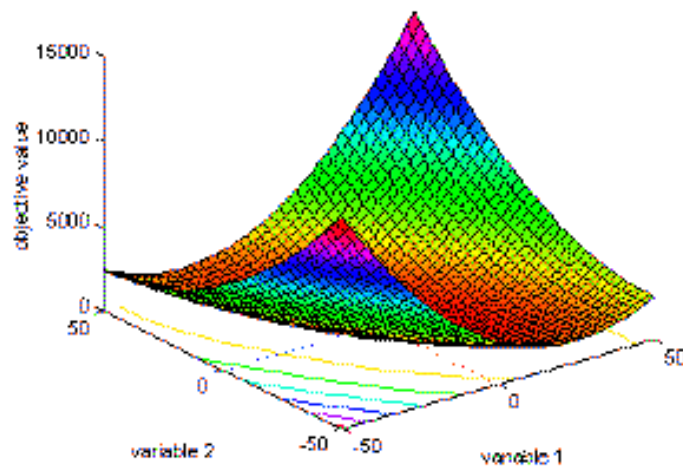


Figure 3.8: Visualization of Rotated hyper-ellipsoid function; surf/mesh plot of the first two variables in an area from -50 to 50. [1]

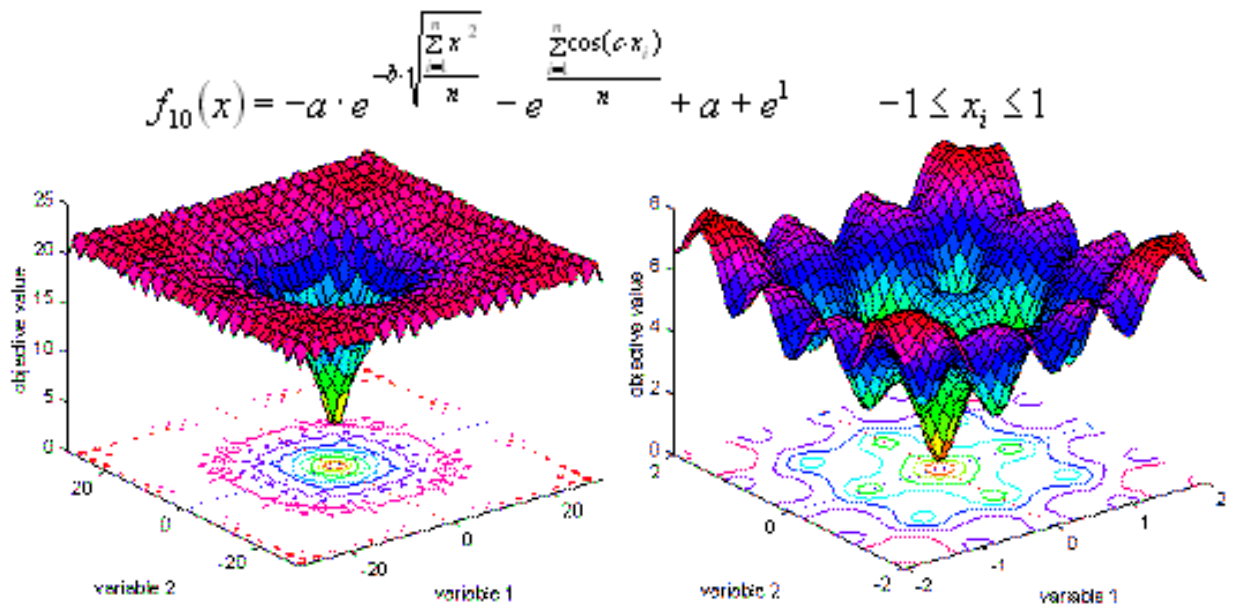


Figure 3.9: Visualization of Ackley's Path function; left: surf plot in an area from -30 to 30, right: focus around the area of the global optimum at $[0, 0]$ in an area from -2 to 2. [1]

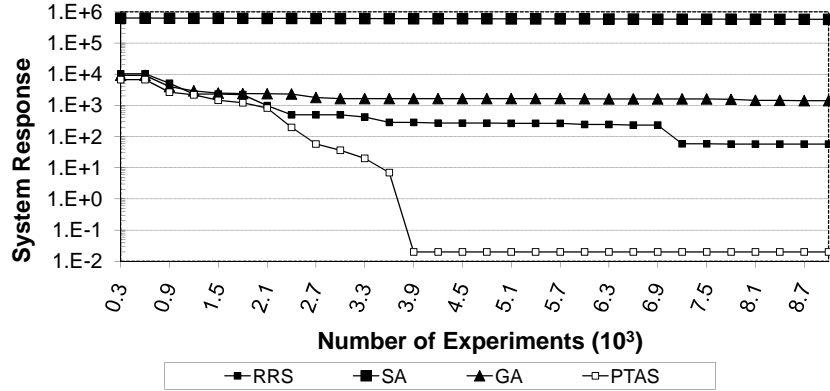


Figure 3.10: Comparison of PTAS with RRS, SA, and GA for SquareSum function.

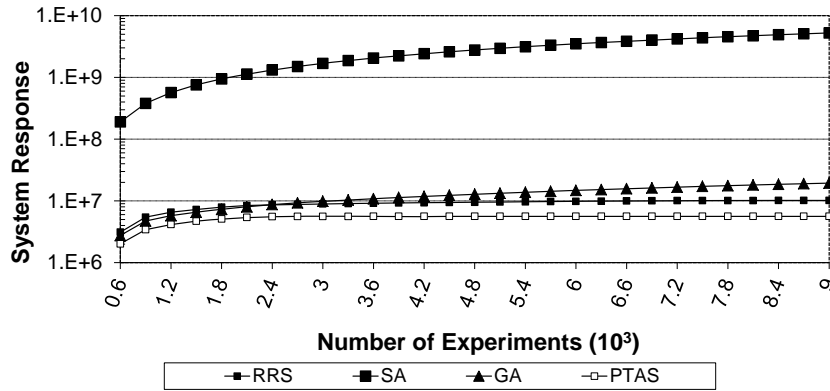


Figure 3.11: Cumulative progresses of four search algorithms for SquareSum function.

3.3.2 When System Response is Fixed

To compare the performance of PTAS with other search algorithms, we experiment with minimizing six different benchmark objective functions. In these experiments, the objective function corresponding to the black-box system does not change. The total budgets in our experiments are 9000. We run each experiment five times with different seed numbers and use the average of five runs as the best-so-far value.

Figure 3.10 shows a comparison of the four algorithms when the objective function is the SquareSum function. In the SquareSum function, because there is only one global optimum and no local optima, PTAS significantly outperformed the

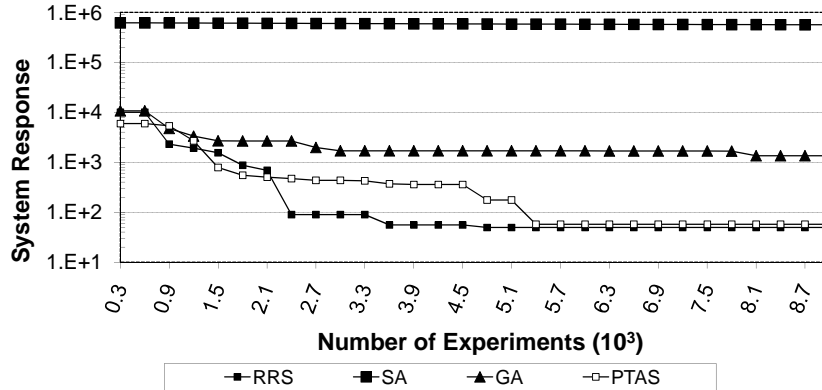


Figure 3.12: Comparison of PTAS with RRS, SA, and GA for Rastrigin function.

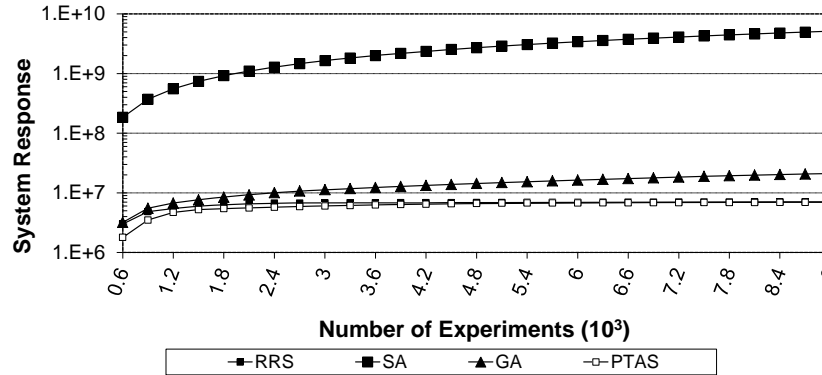


Figure 3.13: Cumulative progresses of four search algorithms for Rastrigin function.

three individual algorithms by large margins. The global optimum point is zero and PTAS almost got there at the 3900th experiment. Figure 3.11 shows the total area that is below the curves in Figure 3.10. According to this graph, the PTAS also outperforms when we consider the cumulative progress for the entire experiment duration. The improvement of the SA is not so obvious, because we used logarithmic scale on the Y-axis.

Figure 3.12 shows a comparison of the four algorithms when the objective function is the Rastrigin function. In the Rastrigin objective function, there are too many local optimum points. Since RRS is a good algorithm in exploration, it

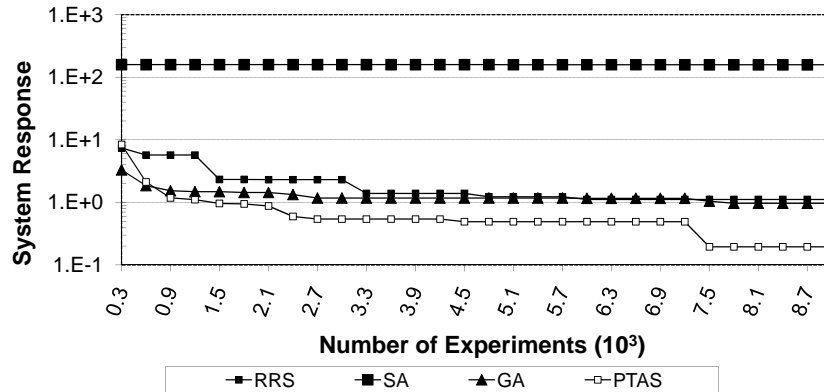


Figure 3.14: Comparison of PTAS with RRS, SA, and GA for Griewangk function.

outperformed Genetic Algorithm and Simulated Annealing algorithms. In this case, until the 2100th experiment, PTAS wins for most cases we examined. Then between the 2100th and 5400th experiment, RRS wins. Then at the 5400th experiment, PTAS caught up with RRS. Then PTAS becomes the second best after RRS with a very small margin. Figure 3.13 shows the total area that is below the curves in Figure 3.12. Although PTAS becomes the second best after RRS with a very small margin at the end of the experiment, PTAS outperforms RRS when we consider the cumulative progress for the entire experiment duration. The improvement of the SA is not so obvious, because we used logarithmic scale on the Y-axis.

Figure 3.14 shows a comparison of the four algorithms when the objective function is the Griewangk function. The Griewangk objective function has many local optimum points. Since RRS and Genetic Algorithm are good algorithms in exploration, they performed better than simulated annealing. Because PTAS uses all three algorithms, PTAS outperforms RRS and Genetic Algorithm with the help of Simulated Annealing. Figure 3.15 shows the total area that is below the curves in Figure 3.14. According to this graph, the PTAS also performs the best when we con-

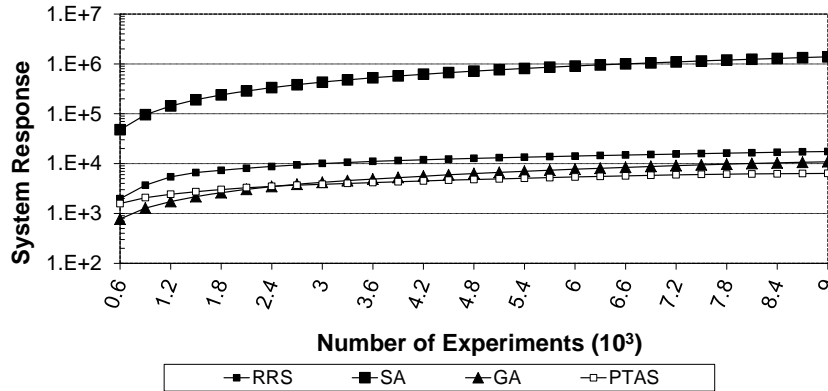


Figure 3.15: Cumulative progresses of four search algorithms for Griewangk function.

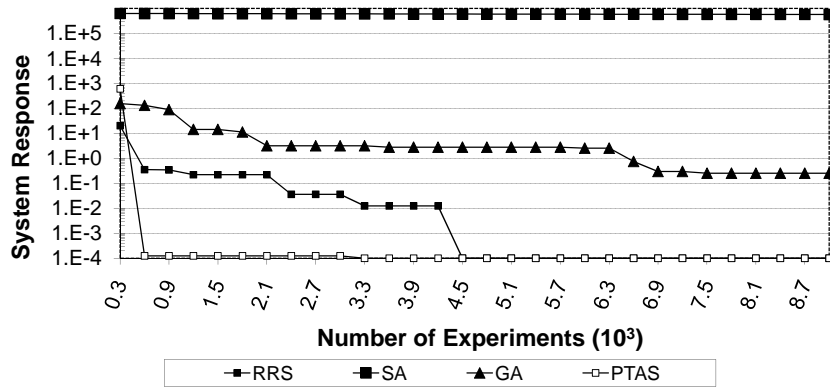


Figure 3.16: Comparison of PTAS with RRS, SA, and GA for Axis parallel hyper-ellipsoid function.

sider the cumulative progress for the entire experiment duration. The improvement of the SA is not so obvious, because we used logarithmic scale on the Y-axis.

Figure 3.16 shows a comparison of the four algorithms when the objective function is the Axis parallel hyper-ellipsoid function. In the Axis parallel hyper-ellipsoid objective function, the global optimum point is zero. PTAS and RRS almost reached zero at the 600th experiment, while GA almost reached it at the 1200th experiment. Until around the 4500th experiment, PTAS wins, and then it was caught later by RRS. Figure 3.17 shows the total area that is below the curves in Figure

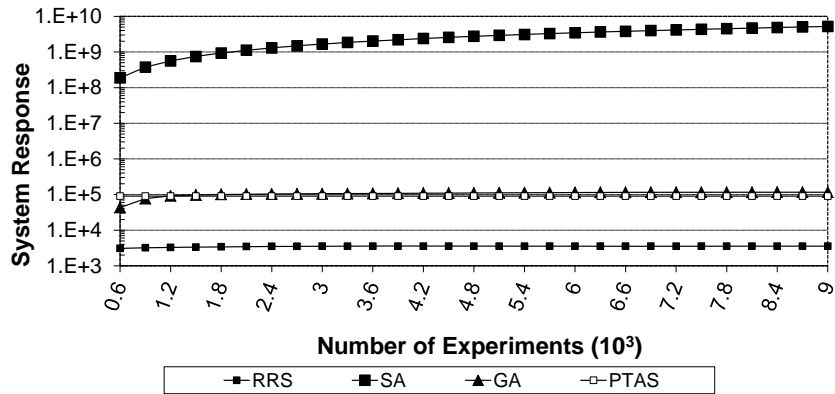


Figure 3.17: Cumulative progresses of four search algorithms for Axis parallel hyper-ellipsoid function.

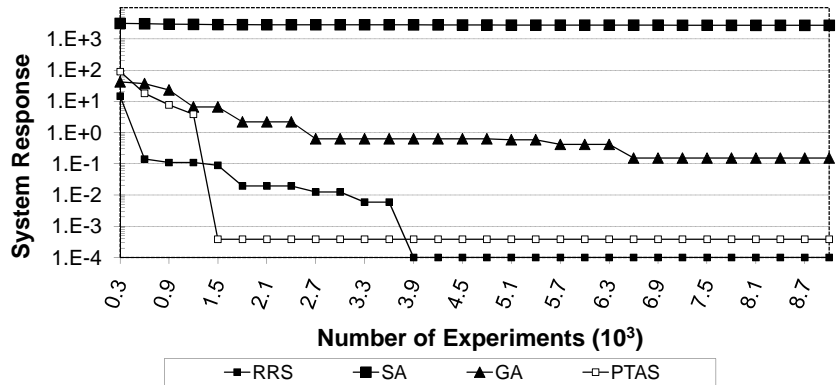


Figure 3.18: Comparison of PTAS with RRS, SA, and GA for Rotated hyper-ellipsoid function.

3.16. According to this graph, when we consider the cumulative progress for the entire experiment duration, RRS performs the best, and PTAS performs the second best. The improvement of the SA is not so obvious, because we used logarithmic scale on the Y-axis.

Figure 3.18 shows a comparison of the four algorithms when the objective function is the Rotated hyper-ellipsoid function. In the Rotated hyper-ellipsoid objective function, the global optimum point is zero. Until the 1200th experiment, RRS

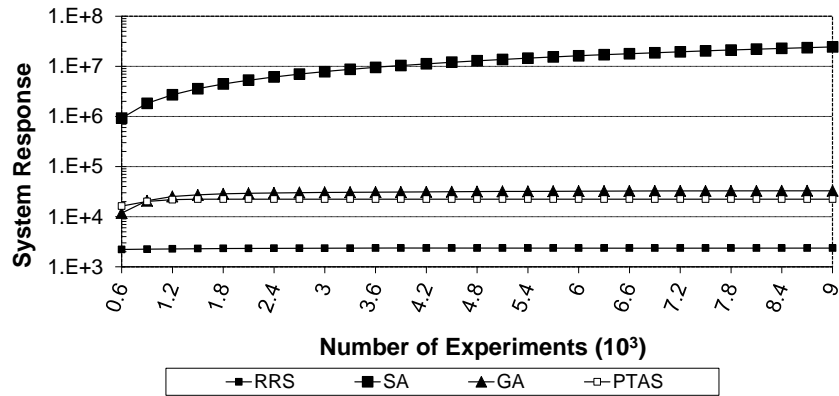


Figure 3.19: Cumulative progresses of four search algorithms for Rotated hyper-ellipsoid function.

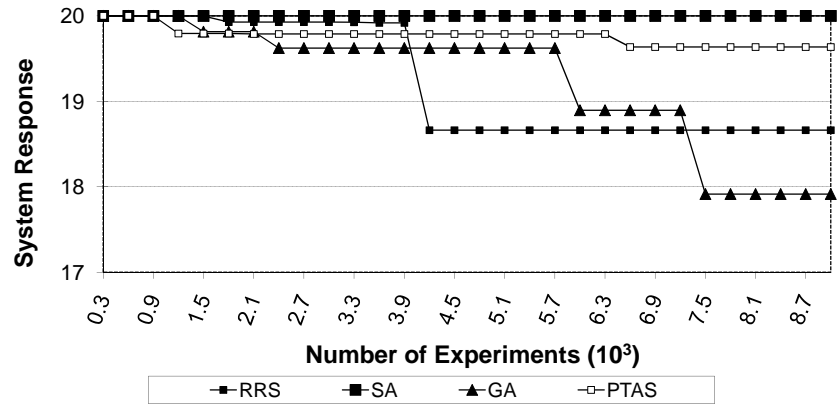


Figure 3.20: Comparison of PTAS with RRS, SA, and GA for Ackley's Path function.

wins, and PTAS remains second. Then until around 3900th experiment, PTAS wins. After the 3900th experiment, RRS passes the PTAS by only a small margin, and PTAS remains second best. Figure 3.19 shows the total area that is below the curves in Figure 3.18. According to this graph, when we consider the cumulative progress for the entire experiment duration, RRS performs the best, and PTAS performs the second best. The improvement of the SA is not so obvious, because we used logarithmic scale on the Y-axis.

Figure 3.20 shows a comparison of the four algorithms when the objective

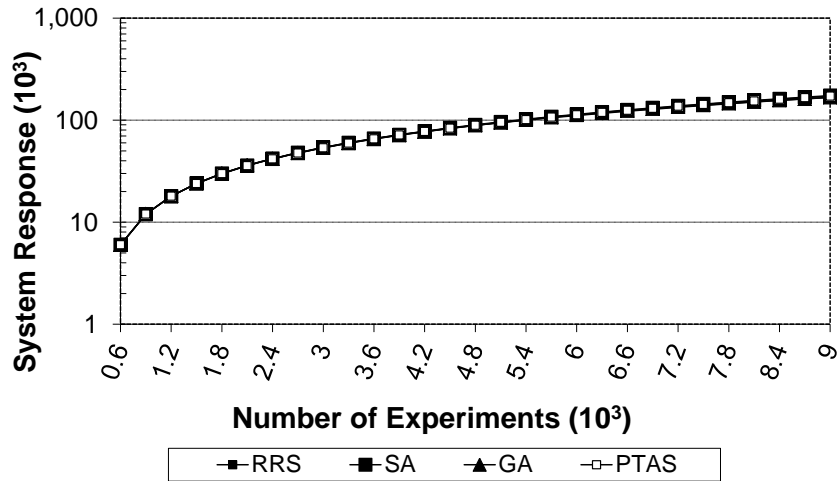


Figure 3.21: Cumulative progresses of four search algorithms for Ackley's Path function.

function is Ackley's Path function. In the Ackley's Path objective function, RRS and Genetic Algorithm gets the first and second places, while PTAS remains third most of the time. Figure 3.21 shows the total area that is below the curves in Figure 3.20. According to this graph, when we consider the cumulative progress for the entire experiment duration, RRS and Genetic Algorithm gets the first and second places, PTAS gets the third place. The improvement of the SA is not so obvious, because we used logarithmic scale on the Y-axis.

For three out of the six different objective functions we tested, PTAS significantly outperformed the three individual algorithms by large margins. For the other three functions (i.e., Ackley, Rotated hyper-ellipsoid, and Rastrigin), PTAS was only slightly outperformed temporarily which suggests that fine tuning of PTAS can still result in better performance. Further, for those three functions, PTAS still performed better early on in the search (i.e., up to 3000 experiments), but was caught up by RRS later. This still makes PTAS more valuable, as the solutions are needed quickly for most real systems.

When we consider the cumulative progress for the entire experiment duration, for three out of the six different objective functions we tested, PTAS performs best. For the two out of six objective functions, PTAS performs second best. And for one out of six objective functions, PTAS gets the third place.

3.3.3 When System Response Varies

In order to evaluate PTAS in a more realistic scenario, we change the objective function in the black-box during the optimization search process. This is a typical situation for large-scale real systems where parts of the system could temporarily fail and the system recovers from such failures. To simulate this, we change the objective function during the optimization search and return back to the original objective function. In large-scale physical systems, small or drastic changes may occur. We have experimented with both cases to illustrate how our PTAS handles drastic (Figures 3.22- 3.31) and small changes (Figures 3.32- 3.37). To show how PTAS performs when there is a drastic change, we change the objective function completely, such as from *SquareSumfunction* to *Rastrigin*, and then to a *SquareSumfunction*. To show how it behaves when there is a small change, we shift the objective function, for example from *Rastrigin* to *Rastrigin - SHIFTED*, and then back to *Rastrigin*. To shift an objective function, we subtract a constant number from each parameter. For instance, we obtained the “shifted” version of the Square Sum function as follows:

$$f_{SquareSum}(x) = \sum_{i=0}^n x_i^2 \quad \text{Global min: } f(x) = 0, x_i = 0, i = 1 : n.$$

$$f_{SquareSum-SHIFTED}(x) = \sum_{i=0}^n (x_i - C)^2 \quad \text{Global min: } f(x) = 0, x_i = C, i = 1 : n.$$

We, again, compare PTAS with the three individual algorithms RRS [56], SA [28], and GA [18]. The total budget in our experiments is 9000. All of the optimization processes in these experiments are minimizations. We run each experiment five times with different seed numbers, and use the average of five runs as the best-so-far values. In each run, an initial objective function is used for the first 3000 experiments, a temporary objective function is used for the next 3000 experiments, and finally, the initial objective function is used again for the third 3000 experiments.

Figure 3.22 shows a comparison of the four algorithms when the objective

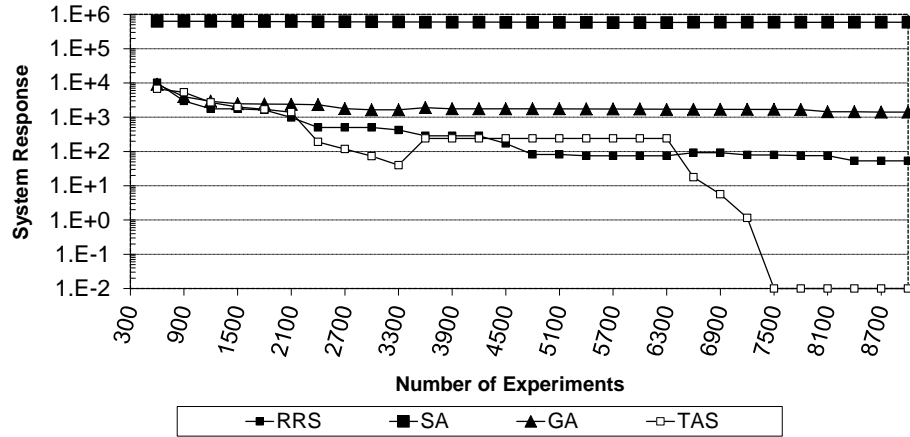


Figure 3.22: A Drastic Change: SquareSum \rightarrow Rastrigin \rightarrow SquareSum

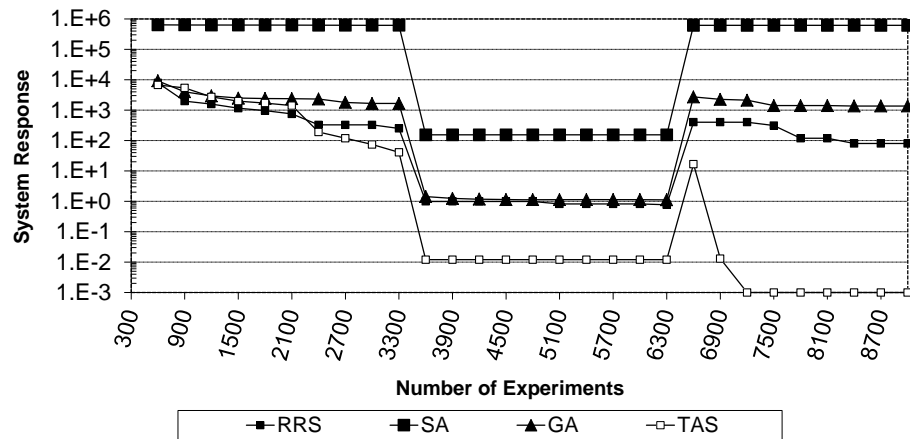


Figure 3.23: A Drastic Change: SquareSum \rightarrow Griewangk \rightarrow SquareSum.

functions are SquareSum, Rastrigin, and then SquareSum. As we observe from our previous experiments (Figure 3.10), PTAS outperforms the SquareSum function. In this case, because the first and third 3000 budget are SquareSum functions, the PTAS outperforms as expected. In the second 3000 budget, RRS wins and PTAS gets the second place. These results are also consistent with the results of our previous experiments (Figure 3.12).

Figure 3.23 shows a comparison of the four algorithms when the objective functions are SquareSum, Griewangk, and then SquareSum. As we observe from

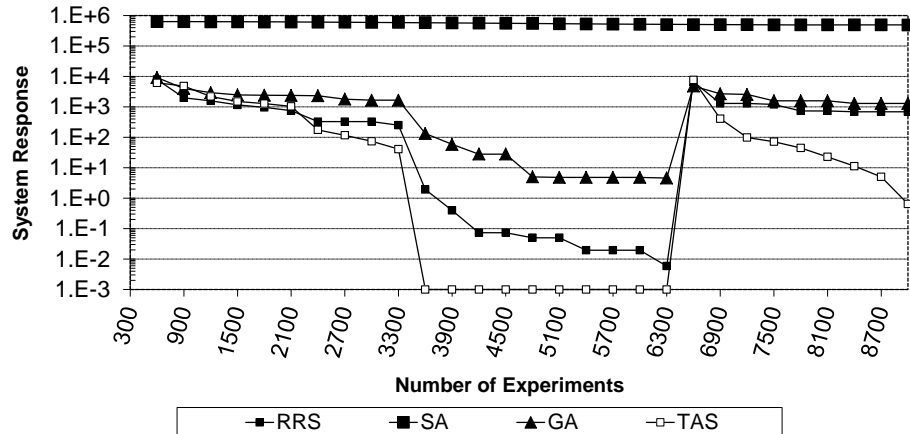


Figure 3.24: A Drastic Change: SquareSum \rightarrow Axis parallel \rightarrow SquareSum.

our previous experiments (Figure 3.10), PTAS outperforms the SquareSum function. Because the first and third 3000 budget are SquareSum functions, PTAS outperforms. Also from our experiments of Figure 3.14, we observe that PTAS outperforms the Griewangk function. Because the objective function is the Griewangk function in the second 3000 budget, PTAS outperforms in that period. These results are consistent with the results of our previous experiments.

Figure 3.24 shows a comparison of the four algorithms when the objective functions are SquareSum, Axis parallel, and then SquareSum. As we observe from our experiments of Figure 3.10, PTAS outperforms for the SquareSum function. In this case, because the first and third 3000 budget is SquareSum function, the PTAS outperforms. Also from our experiments of Figure 3.16, we observe that PTAS outperforms in Axis parallel function. Because the objective function is Axis parallel function in the second 3000 budget, PTAS outperforms in that period. These results are consistent with the results of our previous experiments.

Figure 3.25 shows a comparison of the four algorithms when the objective functions are Rastrigin, SquareSum, and then Rastrigin. As we observe from our

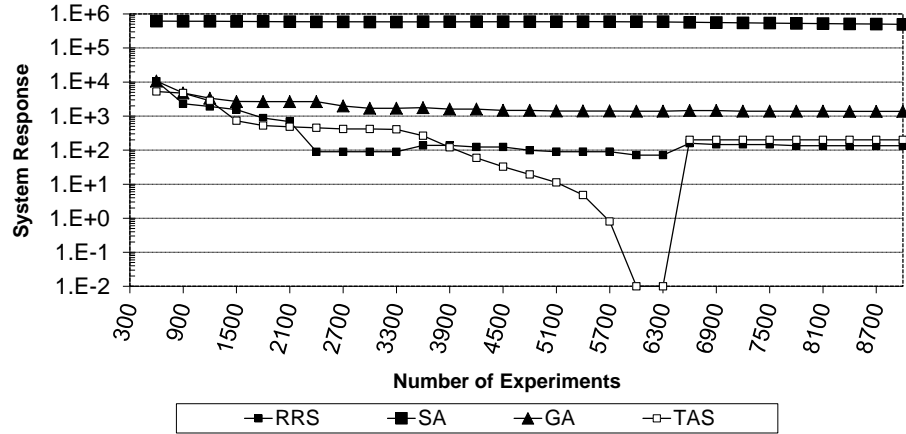


Figure 3.25: A Drastic Change: Rastrigin \rightarrow SquareSum \rightarrow Rastrigin.

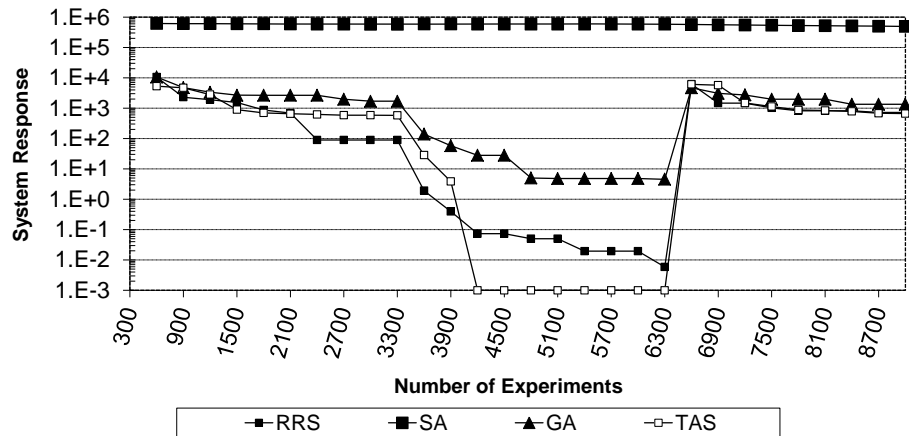


Figure 3.26: A Drastic Change: Rastrigin \rightarrow Axis parallel \rightarrow Rastrigin.

previous experiments (Figure 3.12), RRS outperforms for the Rastrigin function. In this case, because the first and third 3000 budgets are Rastrigin function, the RRS gets the first place, and PTAS gets the second place. Because the objective function is SquareSum function in the second 3000 budget, PTAS outperforms in that period. These results are also consistent with the results of our experiments of Figure 3.10.

Figure 3.26 shows a comparison of the four algorithms when the objective functions are Rastrigin, Axis parallel, and then Rastrigin. As we observe from our experiments of Figure 3.12, RRS outperforms the Rastrigin function. In this case,

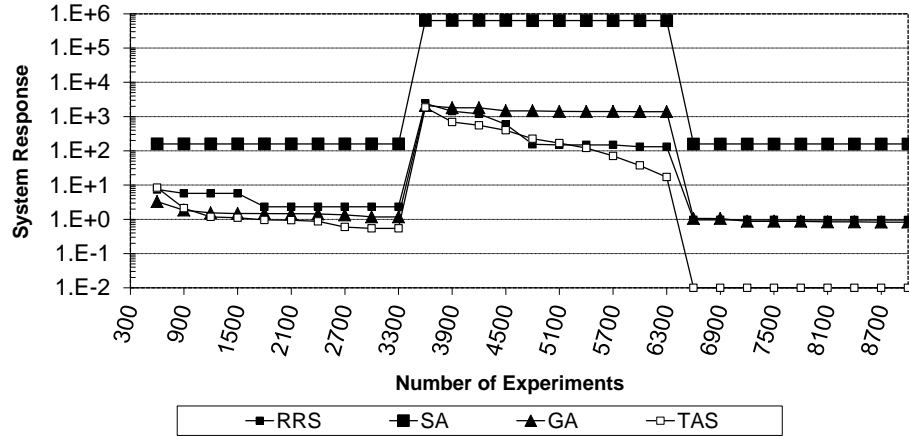


Figure 3.27: A Drastic Change: Griewangk \rightarrow SquareSum \rightarrow Griewangk.

because the first 3000 budget is Rastrigin function, the RRS gets the first place, and PTAS gets the second place. But, in third 3000 budget, although the objective function is Rastrigin function, PTAS gets the first place, and RRS gets the second place. Because the objective function is Axis parallel function in the second 3000 budget, PTAS outperforms in that period. These results are also consistent with the results of the experiments of Figure 3.16.

Figure 3.27 shows a comparison of the four algorithms when the objective functions are Griewangk, SquareSum, and then Griewangk. As we observe from the experiments of Figure 3.14, PTAS outperforms for the Griewangk function. In this case, because the first and third 3000 budgets are Griewangk function, the PTAS gets the first place. Also from our experiments of Figure 3.10, we observe that PTAS outperforms in SquareSum function. Because the objective function is SquareSum function in the second 3000 budget, PTAS outperforms in that period. These results show consistency with the results of our previous experiments.

Figure 3.28 shows a comparison of the four algorithms when the objective functions are Griewangk, Rastrigin, and then Griewangk. As we observe from our

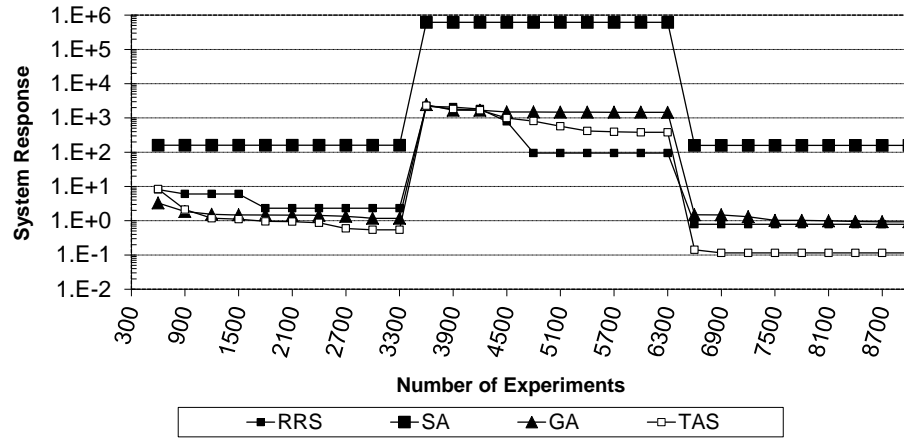


Figure 3.28: A Drastic Change: Griewangk's \rightarrow Rastrigin \rightarrow Griewangk's

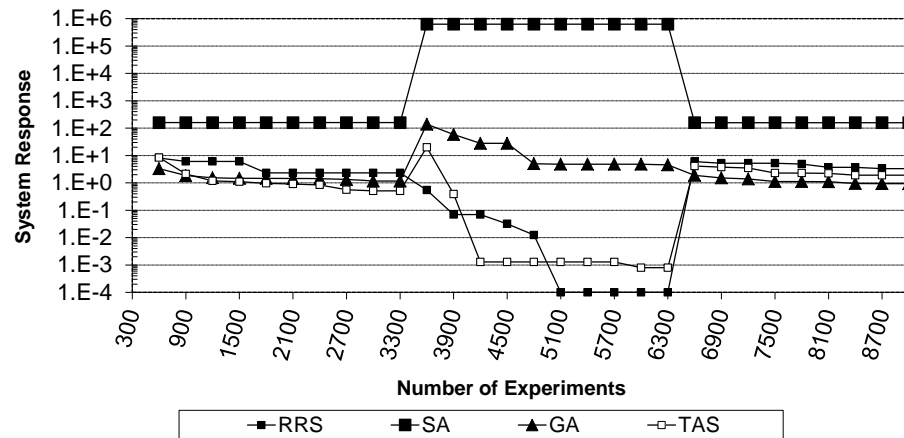


Figure 3.29: A Drastic Change: Griewangk \rightarrow Axis parallel \rightarrow Griewangk.

previous experiments (Figure 3.14), PTAS outperforms for the Griewangk function. In this case, because the first and third 3000 budgets are Griewangk function, the PTAS gets the first place. Also from our experiments of Figure 3.12, we observe that RRS outperforms in Rastrigin function. Because the objective function is Rastrigin function in the second 3000 budget, RRS outperforms in that period. These results show consistency with the results of our previous experiments.

Figure 3.29 shows a comparison of the four algorithms when the objective functions are Griewangk, Axis parallel, and then Griewangk. As we observe from our

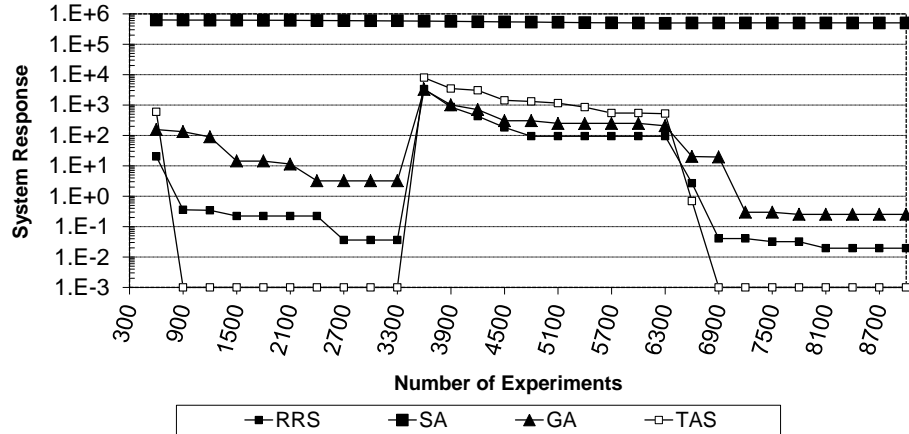


Figure 3.30: A Drastic Change: Axis parallel \rightarrow Rastrigin \rightarrow Axis parallel.

previous experiments (Figure 3.14), PTAS outperforms for the Griewangk function. In this case, because the first 3000 budget is Griewangk function, the PTAS gets the first place. In third 3000 budget, although the objective function is Griewangk function, the GA gets the first place, and PTAS gets the second place. The objective function is Axis parallel function in the second 3000 budget. From our experiments of Figure 3.16, we observe that PTAS outperforms in Axis parallel function. In this case however, PTAS performed well at the beginning of this period, but then RRS outperforms PTAS with a small margin.

Figure 3.30 shows a comparison of the four algorithms when the objective functions are Axis parallel, Rastrigin, and then Axis parallel. As we observe from the experiments of Figure 3.16, PTAS outperforms for the Axis parallel function. In this case, because the first and third 3000 budgets are the Axis parallel function, PTAS gets the first place. Also from the experiments of Figure 3.12, we observe that RRS outperforms in the Rastrigin function. Because the objective function is the Rastrigin function in the second 3000 budget, RRS outperforms in that period. These results show consistency with the results of our previous experiments.

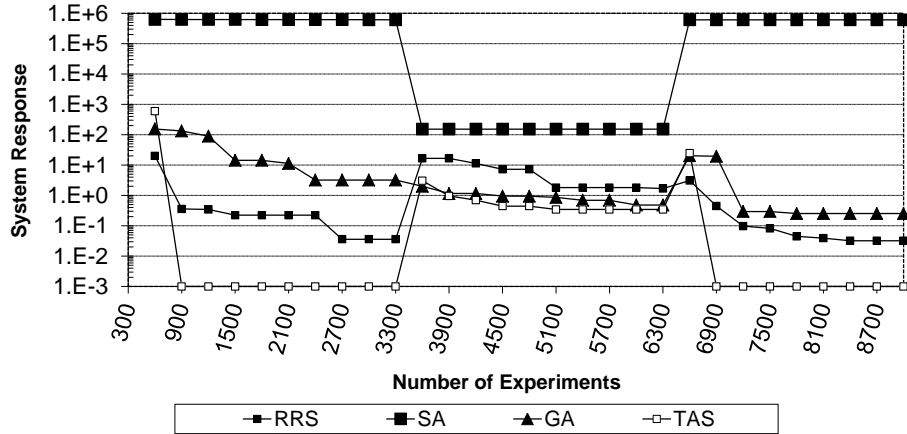


Figure 3.31: A Drastic Change: Axis parallel \rightarrow Griewangk \rightarrow Axis parallel.

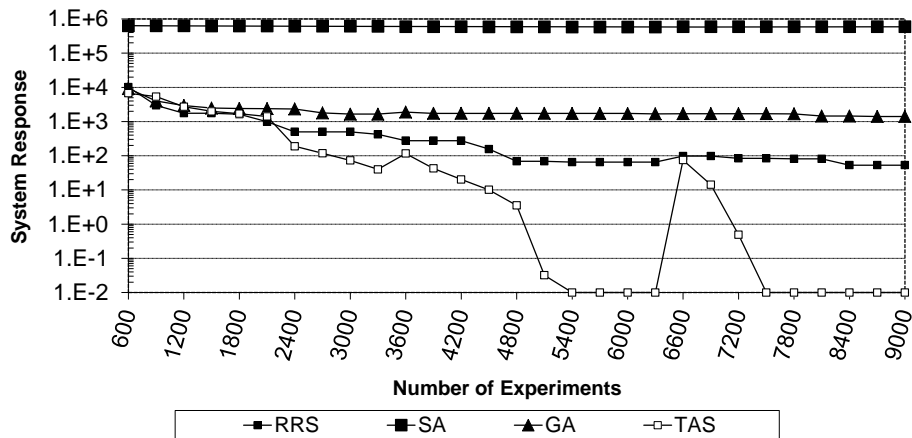


Figure 3.32: A Small Change: SquareSum \rightarrow SquareSum-Shifted \rightarrow SquareSum.

Figure 3.31 shows a comparison of the four algorithms when the objective functions are Axis parallel, Griewangk, and then Axis parallel. As we observe from our experiments of Figure 3.16, PTAS outperforms for the Axis parallel function. In this case, because the first and third 3000 budgets are Axis parallel function, the PTAS gets the first place. Also from our experiments of Figure 3.14, we observe that PTAS outperforms in the Griewangk function. Because the objective function is the Griewangk function in the second 3000 budget, PTAS outperforms in that period. These results show consistency with the results of our previous experiments.

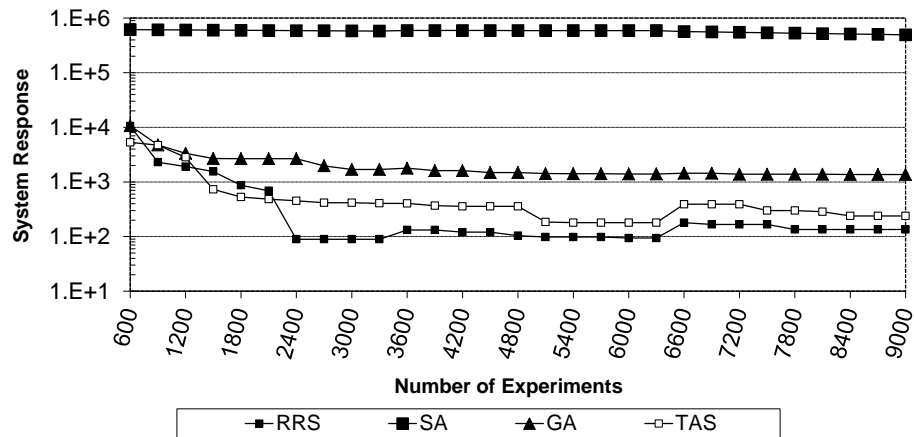


Figure 3.33: A Small Change: Rastrigin \rightarrow Rastrigin-Shifted \rightarrow Rastrigin.

Figure 3.32 shows a comparison of the four algorithms when the objective functions are SquareSum, SquareSum-Shifted, and then SquareSum. As we observe from our experiments of Figure 3.10, PTAS outperforms for the SquareSum function. As expected, PTAS gets the first places in all three 3000 budget periods. These results are consistent with the results of the experiments of Figure 3.10.

Figure 3.33 shows a comparison of the four algorithms when the objective functions are Rastrigin, Rastrigin-Shifted, and then Rastrigin. As we observe from our experiments of Figure 3.12, RRS outperforms for the Rastrigin function. As expected, RRS gets the first places in all three 3000 budget periods, and PTAS gets the second places. These results are also consistent with the results of our experiments of Figure 3.12.

Figure 3.34 shows a comparison of the four algorithms when the objective functions are Griewangk, Griewangk-Shifted, and then Griewangk. As we observe from our experiments of Figure 3.14, PTAS outperforms for the Griewangk function. As expected, PTAS gets the first places in all three 3000 budget periods. These results are consistent with the results of our experiments of Figure 3.14.

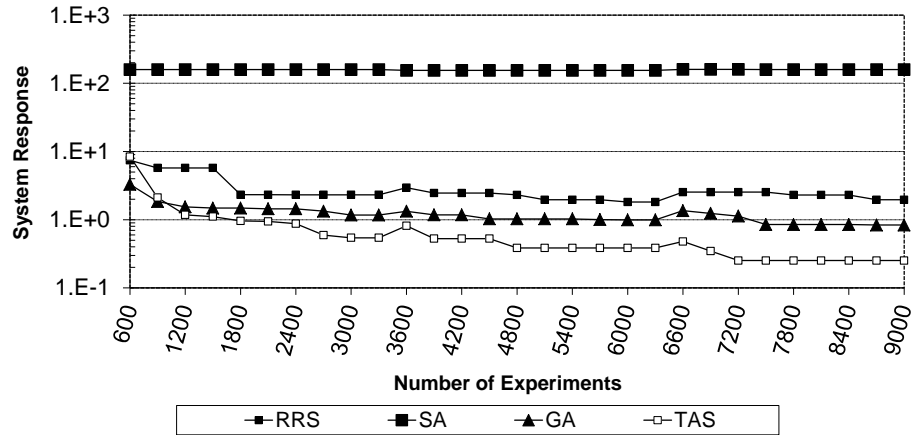


Figure 3.34: A Small Change: Griewangk \rightarrow Griewangk-Shifted \rightarrow Griewangk.

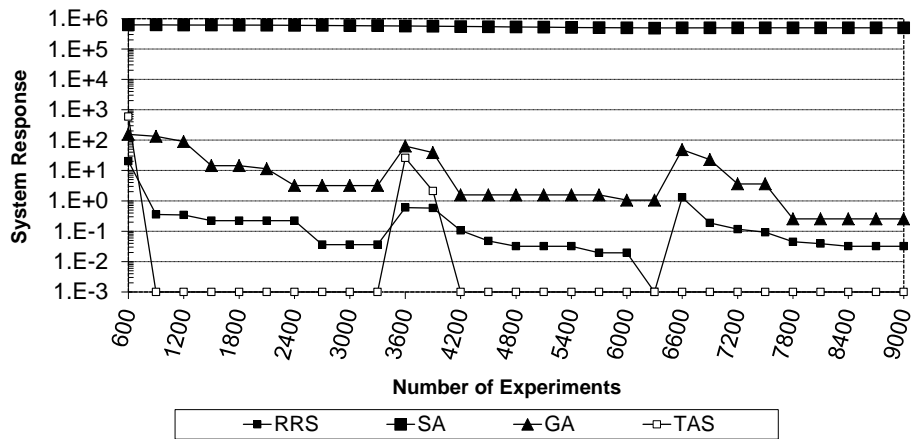


Figure 3.35: A Small Change: Axis parallel \rightarrow Axis parallel-Shifted \rightarrow Axis parallel.

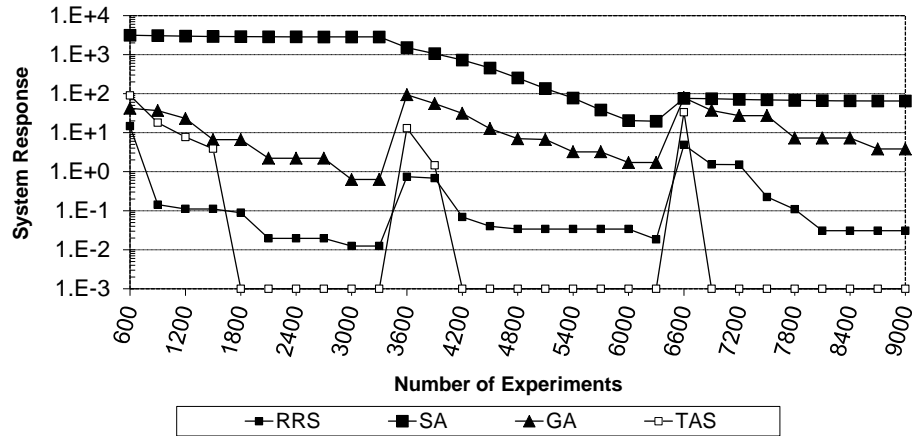


Figure 3.36: A Small Change: Rotated ellipsoid \rightarrow R. ellipsoid-Shifted \rightarrow R. ellipsoid.

Figure 3.35 shows a comparison of the four algorithms when the objective functions are Axis parallel, Axis parallel-Shifted, and then Axis parallel. As we observe from our experiments of Figure 3.16, PTAS outperforms for the Axis parallel function. As expected, PTAS gets the first places in all three 3000 budget periods. Actually, just after the search space is shifted, PTAS becomes second best, but then it quickly adapts, and becomes the first place in around 600 experiments. These results are consistent with the results of our experiments of Figure 3.16.

Figure 3.36 shows a comparison of the four algorithms when the objective functions are Rotated ellipsoid, Rotated ellipsoid-Shifted, and then Rotated ellipsoid. In our experiments of Figure 3.18, we observe that PTAS outperforms for the Rotated ellipsoid function at the beginning, then RRS passes PTAS when experiments run long enough. In this scenario, PTAS gets the first places in all three 3000 budget periods. Actually, just after the search space is shifted in both second and third 3000 budget periods, PTAS becomes second best, then it quickly adapts, and becomes the first place in around 300 experiments. These results are consistent with the results of our experiments of Figure 3.18.

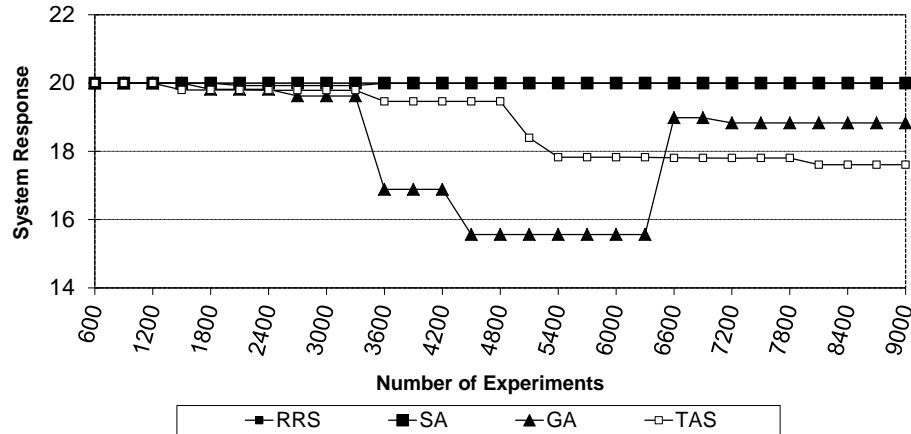


Figure 3.37: A Small Change: Ackley's \rightarrow Ackley's-Shifted \rightarrow Ackley's.

Figure 3.37 shows a comparison of the four algorithms when the objective functions are Ackley's, Ackley's-Shifted, and then Ackley's. In our previous experiments (Figure 3.20), we observe that GA outperforms for the Ackley's function. In this scenario, GA gets the first places in the first and second 3000 budget periods. And these results are consistent with the results of our experiments of Figure 3.20. However, PTAS becomes the best in the third 3000 budget period.

Chapter 4

Online Optimization of Networks with PTAS

As the size, dynamism and diversity of network components are increasing, the problem of finding the best configuration settings for networks is becoming more difficult. Manageability of large-scale networks is highly dependent on tools and methods to configure them. The typical practice has been to train and employ highly-experienced human administrators for network configuration and management. However, automated ways of managing and configuring networks are essential for scaling this practice for highly dynamic, heterogeneous and large networks. Thus, tools and methods to achieve automated management and configuration of a running network are vitally needed.

Tuning and configuring a running network or system requires one to perform “online” optimization by searching and finding better parameter configuration settings of the system at hand. Such optimization of systems can be performed in two ways: (i) *using a separate model of the system* for experimenting with and trying new

configurations, as depicted in Figure 4.1, or (ii) *using the system itself* for experimentation without a separate system model as depicted in Figure 4.2. The former approach is appropriate when the system is steady-state and exhibits the same or similar behavior over long periods of time. Such steady-state systems can be characterized by accurate models. For instance, backbone networks or wireless mesh networks with steady-state links and invariant traffic profile give a fixed response and can be accurately modeled over long time periods. Our first version of PTAS, called *PTAS with separate system model*, explored the optimization of such networks for very hard configuration problems such as Interior Gateway Protocol (IGP) link weight setting for load balancing [58].

For systems with highly dynamic behavior, it is not practical to accurately capture the system’s response with a model. Modeling of the actual system becomes impractical since the system behavior changes frequently. Thus, the approach without a system model is needed for such systems. For large networks with high link/router failure rate or a very dynamic traffic demand profile, or mobile ad-hoc networks (MANETs) with a constantly changing topology, it is a necessity to use the system itself to try out new configurations with the hope of finding a better parameter setting. A key challenge in this particular case is the fact that the objective function changes during the optimization itself. Thus, the optimization process must adapt to the changes in the objective function by applying the appropriate search algorithms for the system response due to the new system dynamics.

In the second version of PTAS, which is called *PTAS with no system model*, we focus on the latter approach and investigate a two-phase online optimization framework for network configuration and management. Our approach follows a pattern of two subsequent phases, *search* and *no-search*, where new configuration parameters

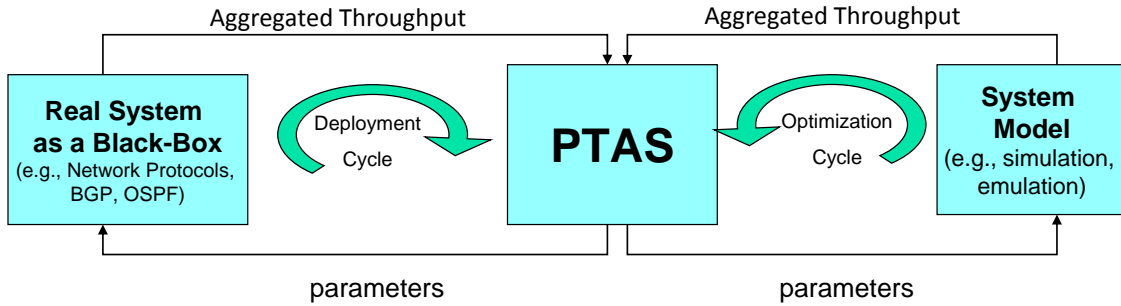


Figure 4.1: Optimization using a separate model of the system

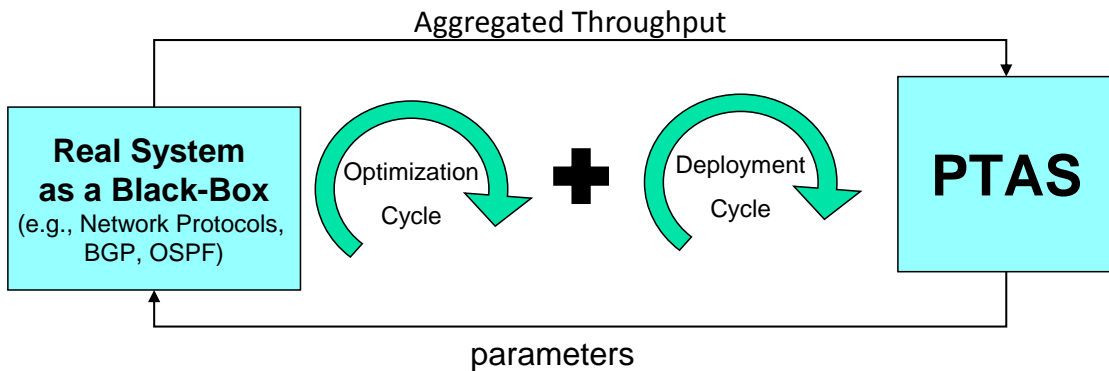


Figure 4.2: Optimization using real-time data from the system

are tried during the search phase. Each search phase is followed by a no-search phase with normal operation using the parameters found in the latest search phase. We explore some of the key tradeoffs in the two-phase model, such as the frequency of the search, the duration of the search phase, and the adverse effect of the search phase on the system performance. We apply the two-phase optimization model on the IGP link weight setting problem and observe the aggregate network throughput against several parameters such as link failure rate, optimization algorithm, and total experiment budget of the search phases.

In chapter 3, we detailed the PTAS framework, and then applied it to some benchmark objective functions. We included some dynamics in the functions to see

how the PTAS adapts itself against the small and drastic changes. In this chapter, we use PTAS to perform online optimization of some sample network problems to see applicability of it.

In Section 4.2, we discuss our experiment results of PTAS with separate system model on two well-known network problems: (i) the problem of interior gateway protocol (IGP) link weights optimization on realistic ISP topologies and (ii) optimization of ad hoc wireless networks by tuning the data rates of traffic sources. Section 4.3 discusses the experiment results of PTAS with no system model on a real-time running interior gateway protocol (IGP) link weights optimization problem.

4.1 Experiment Setup

In order to validate the accuracy and efficiency of our PTAS algorithm, we used Network Simulator 2 (NS-2) [2] as it is the most commonly used network simulator in the field. It is an open source network simulator and there is a vast amount of documentation about it. NS-2 is written in C++. Users write TCL script codes to describe their network topology, to assign weights to the links of the topology and to give the specifics of the simulation, such as simulation duration, bandwidths of the links, type of connection protocol (e.g. TCP, UDP), type of traffic flows (e.g., Constant Bit Rate, FTP), and type of routing model.

Because NS-2 is an open source network simulator, we can modify the code, or add more features to it. Initially, to allow our PTAS algorithm to communicate with the NS-2, we were using a text file between PTAS and NS-2. This worked well, but when we run lots of simulations with longer duration, we had performance issues, since file I/O operations are slow. We decided to convert our PTAS code into an NS-2

agent and integrate it into the NS-2, which eliminated the need for text files. In the TCL script code, we create an instance of the PTAS agent, then bind some variables of the TCL script code with the variables of the PTAS. Through this variable binding, the PTAS and NS-2 simulator can communicate directly. To run multiple simulations at the same time, we used the following arguments for the NS-2 TCL script.

1. `argchoiceAlg`: which algorithm to use [0-4]. (0 is for No-Algorithm. [1-4] is for 4 search algorithms)
2. `arglinkfail`: The index of the link to fail.
3. `argseed`: The seed number for the random number generator.
4. `tcpDuration`: The duration of TCP traffic flows in seconds.
5. `totalbudget`: Represents how many times the algorithms can try different link weights for their searches.
6. `numofrounds`: The number of rounds for PTAS. This is for PTAS only. For other three algorithms, this argument is ignored. This value is, later, used to calculate the round budget for PTAS.
7. `interfailtimeExp`: We used Exponential distribution for the time interval between link failures. This value is the expected value for the Exponential distribution.
8. `repairtimeExp`: We used Exponential distribution for the repair time of a failed link. This value is the expected repair time value for Exponential distribution.
9. `searchinterval`: The time interval between the beginning times of search phases.



Figure 4.3: One rack of compute nodes in the UNR Research Grid [7]

10. `samplinginterval`: During a search phase, an algorithm assigns a different set of weights to the links. This argument represents the time interval between those tries.

To run extensive experiments, we used the University of Nevada, Reno servers, called “The UNR Research Grid”. According to their website (as of September 3rd, 2010) [7], the grid’s specifications are as follows:

- 149 compute nodes, with a total of 720 CPU Cores.
- Combined 1828 Gigabytes of RAM
- 24 Terabyte NAS (Network-attached storage)

We connected to the research grid via SFTP remote connection. We installed NS-2 on the grid and integrated our PTAS agent in it. In the research grid, we were able to submit thousands of jobs at once by using automated scripts. Although there

is no limit for the number of jobs we can submit, there is a limit of 577 for the number of jobs which can run simultaneously. Once some of these 577 jobs complete, then the jobs waiting in the queue are started. Algorithm 3 shows the outline of the script we used for submitting our simulation jobs to the grid.

Algorithm 3 Grid script

```

1: #!/bin/bash
2: #$ -cwd
3: for argchoiceAlg in 0 1 2 3 4 do
4:   for arglinkfail in 0 1 2 3 4 5 6 7 8 9 do
5:     for argseed in 1 2 3 4 5 6 7 8 9 10 do
6:       for tcpDuration in 17280 do
7:         for totalbudget in 360 720 1080 1440 1800 do
8:           for numofrounds in 5 10 15 20 do
9:             for interfailtimeExp in 5 10 20 30 40 50 100 200 400 800 do
10:              for repairtimeExp in 10 do
11:                for searchinterval in 2000 3000 4000 5000 do
12:                  for samplinginterval in 1 do
13:                    echo “#!/bin/bash
14:                      #$ -cwd
15:                      /home/gonenb/ns-allinone-2.35-RC7/ns-2.35/ns  ospfv2.tcl  $arg-
16:                      choiceAlg $arglinkfail $argseed $tcpDuration $totalbudget $nu-
17:                      mofrounds $interfailtimeExp $repairtimeExp $searchinterval
18:                      $samplinginterval” > job-$argchoiceAlg-$arglinkfail-$argseed-
19:                      $tcpDuration-$totalbudget-$numofrounds-$interfailtimeExp-
20:                      $repairtimeExp-$searchinterval-$samplinginterval.sge
21:                      qsub          job-$argchoiceAlg-$arglinkfail-$argseed-$tcpDuration-
22:                      $totalbudget-$numofrounds-$interfailtimeExp-$repairtimeExp-
23:                      $searchinterval-$samplinginterval.sge
24:                    end for
25:                  end for
26:                end for
27:              end for
28:            end for
29:          end for
30:        end for
31:      end for
32:    end for
33:  end for
34: end for

```

4.2 PTAS with Separate System Model

In this section, we discuss our experimental results for PTAS with separate system model on two well-known network problems. In subsection 4.2.1, we describe the problem of interior gateway protocol (IGP) link weight optimization on realistic ISP topologies and present its results. In subsection 4.2.2, we describe the optimization of ad hoc wireless networks by tuning the data rates of traffic sources and also present its results.

4.2.1 IGP Link Weights Optimization

We mapped PTAS to a simulation-based IGP link weights optimization framework. PTAS selects link weight values for links in the simulated topology and feeds them to an NS-2 simulation of IGP routing. The metric to be optimized is the aggregate throughput of the network. Initially, we randomly assigned some weights between (0-100) to the links with a uniform distribution. Then we ran the simulation with those link weights and let TCP traffic flow for 100s. After each such simulation run, we measured the aggregate throughput that was observed at the egress points of the network topology, and sent the throughput value as “the metric” to PTAS. In the next iteration, PTAS changed some of the link weights in the topology, and reran the simulation. By repeating this iterative process up to an experiment budget, PTAS tries to find the IGP link weights closer to the optimum.

To model the network, we used Rocketfuel’s three different topologies [3]. These topologies are:

1. Exodus topology: 22 nodes and 37 links exist (Figure 4.4).
2. Abovenet topology: 22 nodes and 42 links exist (Figure 4.6).

3. Sprint topology: 44 nodes and 83 links exist (Figure 4.8).

Performance Evaluation on Exodus topology

To model the network, we used Rocketfuel’s Exodus topology [3], for which 22 nodes and 37 links exist (Figure 4.4). We used 7 nodes as the edge nodes, and composed $6 \times 7=42$ TCP flows between those edge nodes. As the simulation metric to be returned to the PTAS, we calculated the total number of bytes received at the sink nodes of the TCP flows. We repeated the optimization process 30 times. Figure 4.5 shows the average throughput achieved by each algorithm with 80% confidence intervals. We observe that algorithms improve their link weights as the experiment budget increases. PTAS outperforms the other three algorithms and its comparative performance edge becomes more pronounced as the experiment budget increases.

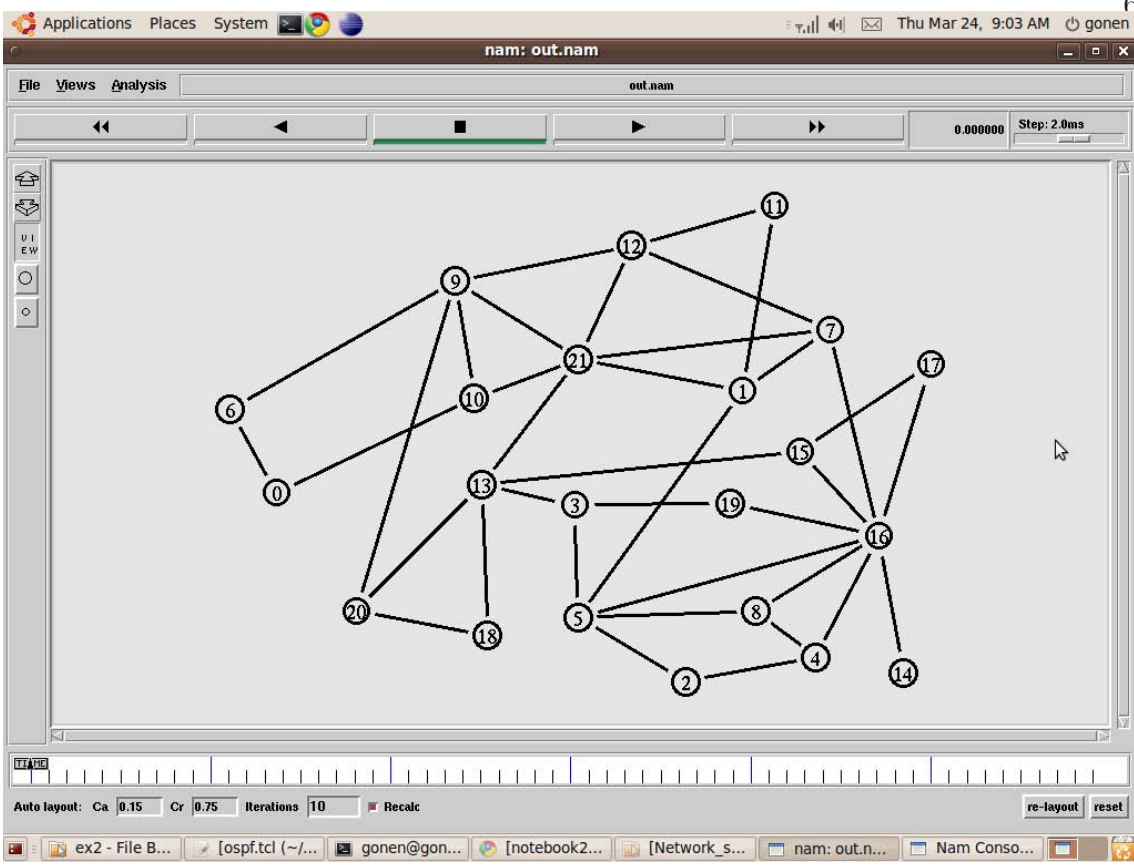


Figure 4.4: Rocketfuel's Exodus topology.

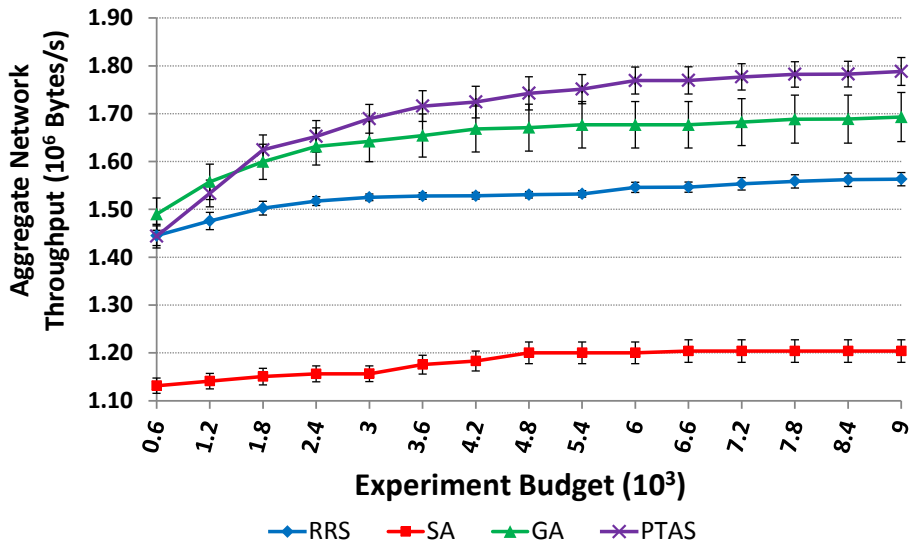


Figure 4.5: Results of experiments on Exodus topology with different experiment budgets.

Performance Evaluation on Abovenet topology

To model the network, we used Rocketfuel's Abovenet topology, for which 22 nodes and 42 links exist (Figure 4.6). In an ISP network topology, usually 80% of the routers are edge nodes and 20% of the routers are core routers. Therefore, we used 18 nodes as the edge nodes, and we established TCP connections from each node to every other node. That is $18 \times 17 = 306$ TCP flows between those edge nodes. As the simulation metric to be returned to the PTAS, we calculated the total number of bytes received at the sink nodes of the TCP flows. We ran simulations for each duration 30 times and the values in the Y-Axis are the average throughput achieved for those 30 runs. The results are very promising. For 4 out of 7 cases, the PTAS found the best result among four search algorithms. For 3 out of 7 runs, the RRS [56] algorithm found the best result, and PTAS found the second best result. We observe that algorithms improve their link weights as the experiment budget increases. PTAS outperforms the other three algorithms and its performance edge becomes more pronounced as the experiment budget increases. Figure 4.7 shows the average throughput achieved by each algorithm with 80% confidence intervals.

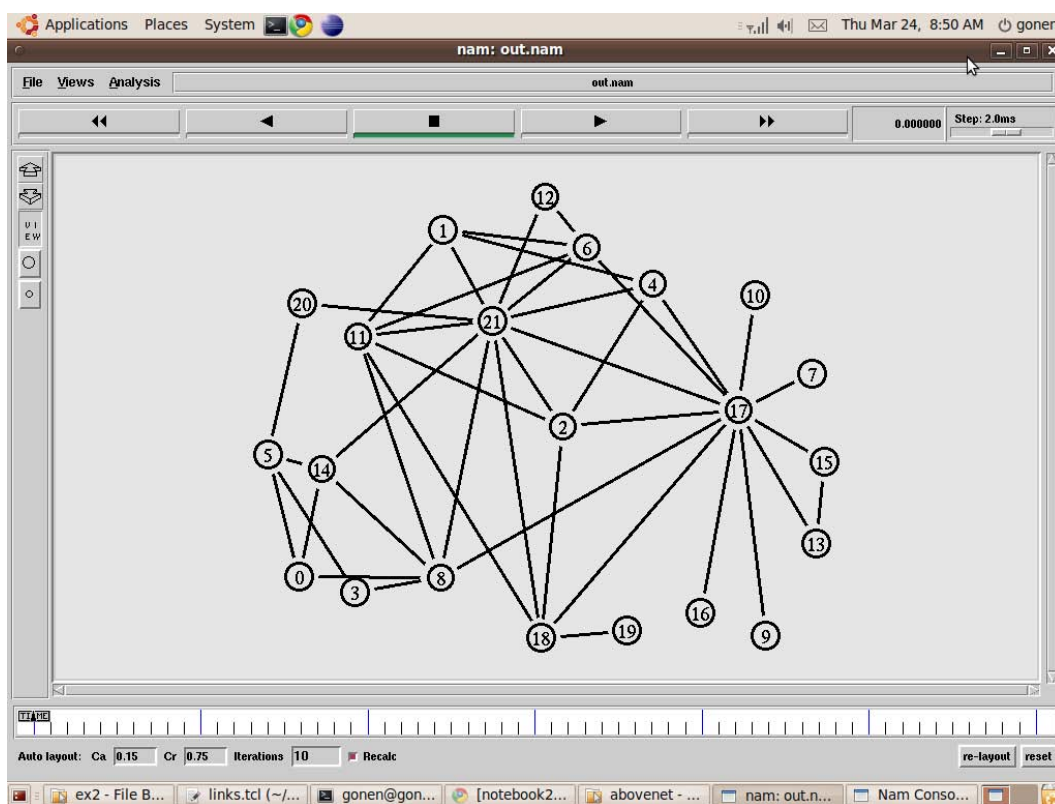


Figure 4.6: Rocketfuel's Abovenet topology

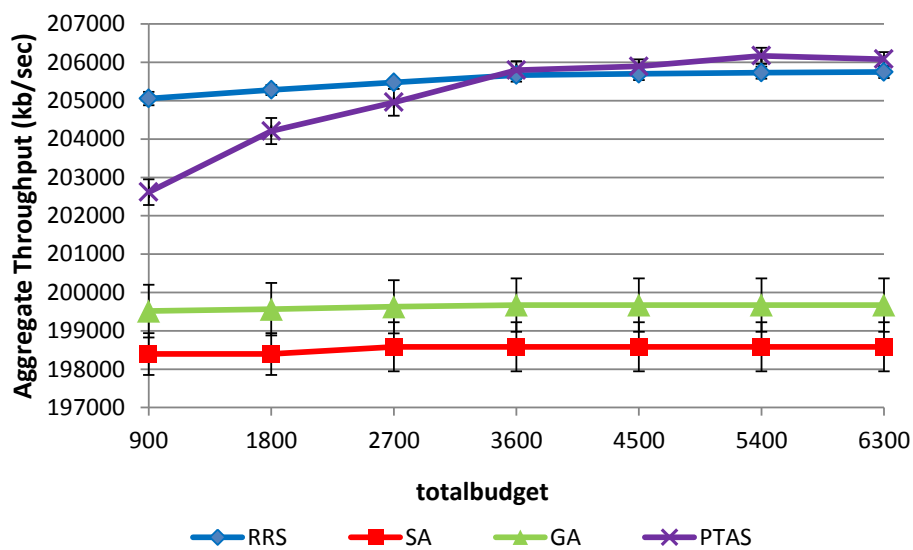


Figure 4.7: Results of experiments on Abovenet topology with different experiment budgets

Performance Evaluation on Sprint topology

To model the network, we used Rocketfuel's Sprint topology, for which 44 nodes and 83 links exist (Figure 4.8). We used 36 nodes as the edge nodes and established TCP connections from each node to every other nodes. That is $36 \times 35 = 1260$ TCP flows between those edge nodes. As the simulation metric to be returned to the PTAS, we calculated the total number of bytes received at sink nodes of the TCP flows. To gain confidence, we run simulations for each duration 10 times. Figure 4.9 shows the performance comparison of the algorithms for each of 10 runs. The numbers in the X-Axis represent the random seed number used in that run. In this graph, the simulation durations are 6300 seconds and as the number of rounds for PTAS, we used four different values, i.e., 5, 10, 15, and 20. Therefore, the values in the Y-Axis are the average of those four simulations for each seed number.

Figure 4.10 shows the performance comparison of the algorithms for different experiment budgets with 80% confidence intervals. The values in the X-Axis represent the number of samplings for each search algorithm. Because our sampling interval time is one second, the total budget also corresponds to simulation durations in seconds. In this graph, we used 10 rounds for PTAS and ran simulations for each duration 10 times. The values in the Y-Axis are the average throughput achieved of those 10 runs. The results are also good for this experiment. PTAS found either the best result, or the second best result among four search algorithms. We also observe that algorithms improve their link weights as the experiment budget increases. The comparative performance edge of PTAS becomes more pronounced as the experiment budget increases.

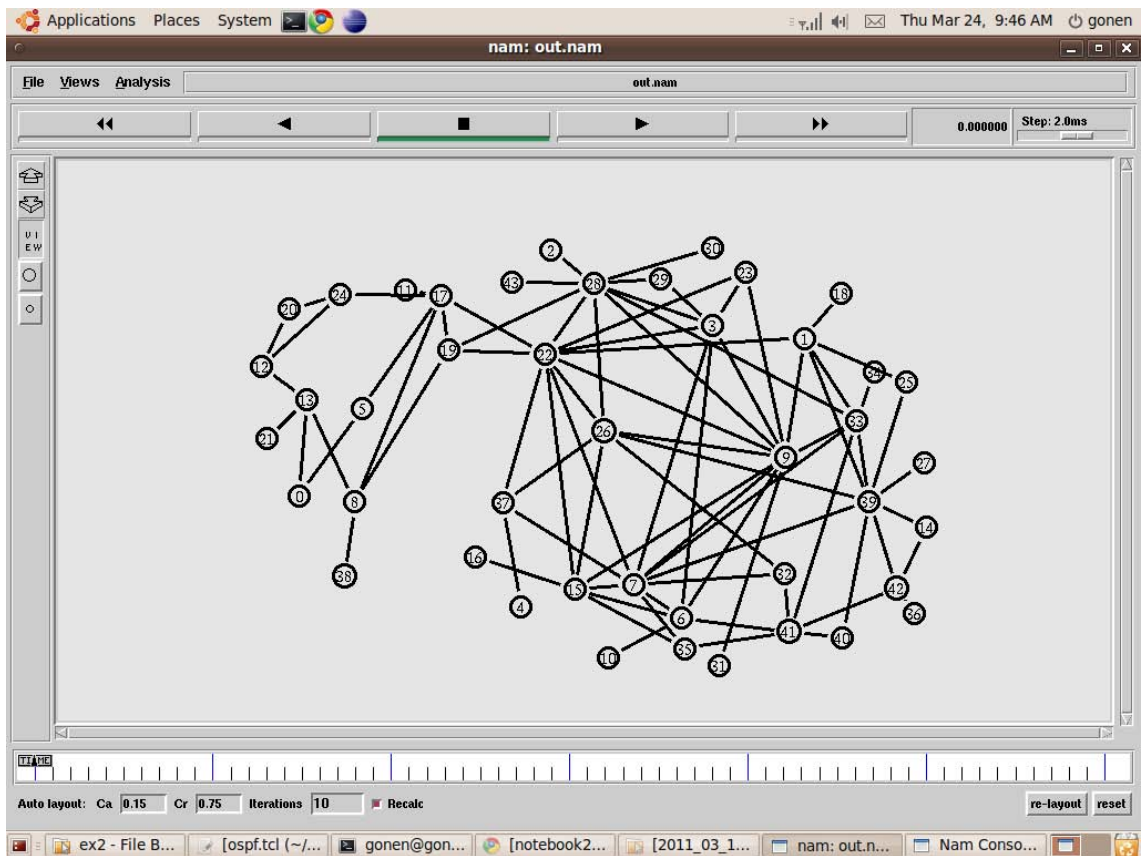


Figure 4.8: Rocketfuel's Sprint topology.

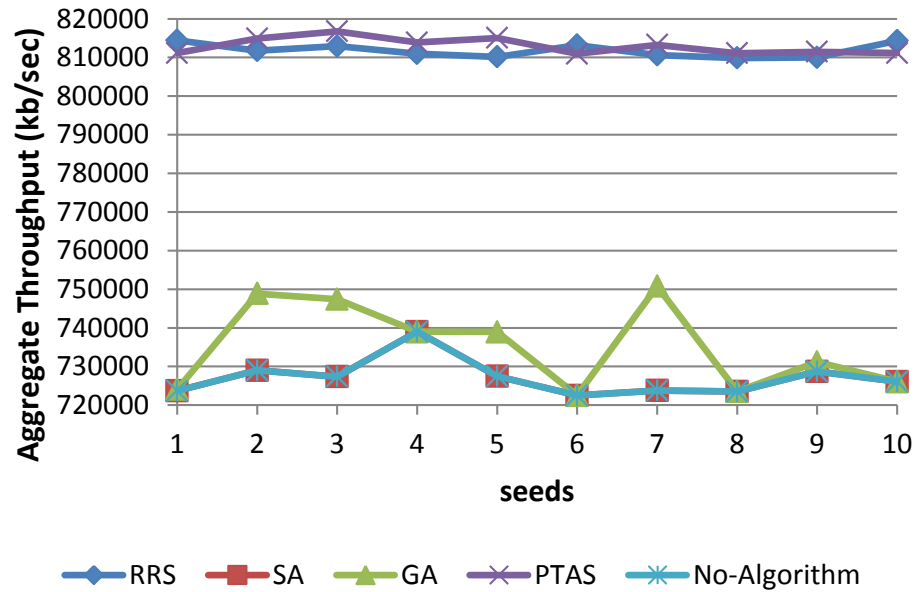


Figure 4.9: Results of experiments on Sprint topology with different random seeds for each run.

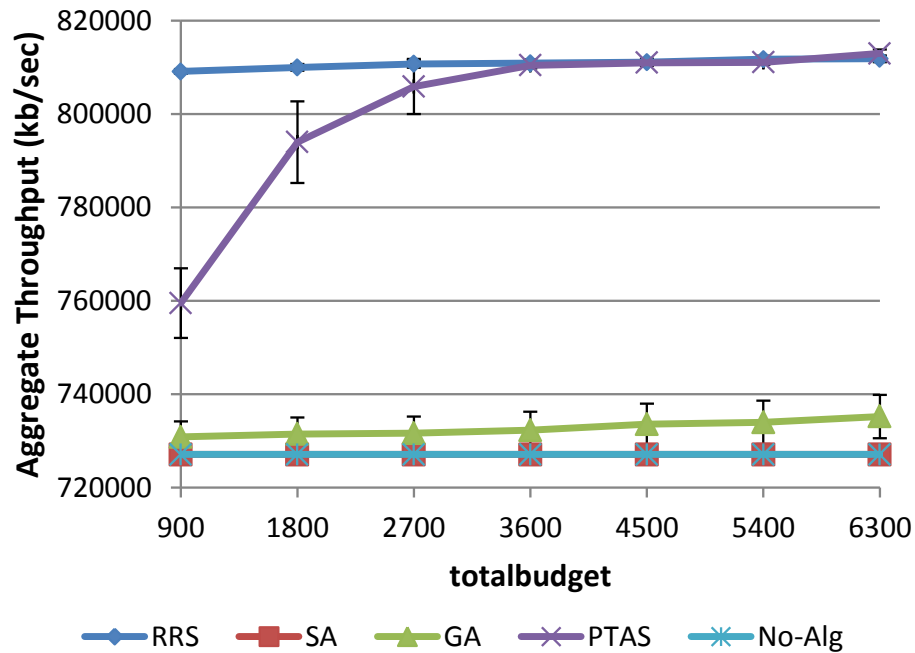


Figure 4.10: Results of experiments on Sprint topology with different experiment budgets.

4.2.2 Optimization of Ad Hoc Networks

In order to compare our PTAS algorithm with other three individual search algorithms, we used NS-2 network simulator [2] to optimize a wireless ad hoc network by tuning data rates of traffic sources. As the output metric to optimize, we use aggregate throughput in the topology. Throughput is defined as the total number of packets received by the destination successfully and is a measure of the effectiveness of a routing protocol [17]. Some of the variables that affect throughput of a wireless data communications system are transmission data rate, packet size, received signal power, received noise power, channel conditions, and modulation technique [5].

Consider a scenario where some of the nodes are in the middle of the network topology and are very close to each other whereas some other nodes are at the edges of the network topology and are far away from most of the nodes. Assume that we use the same transmission rate for every node in the network. If we set the transmission rates sufficiently high for the edge nodes to transmit their data to other nodes, then the middle nodes will experience high interference, thus corrupting the total transmission. On the other hand, if we set the transmission rates sufficiently low to reduce the interference in the middle nodes, then the edge nodes will not be able to transmit their data to other nodes, thus reducing the total transmission. Our goal here is to set optimum transmission data rates for each node in the network that minimizes the packet loss and maximizes the total transmission in the network.

We run our experiments under two different scenarios. In the first scenario, we created 4 nodes in a 700m x 700m area established UDP connections from each node to every other node, i.e., $4 \times 3 = 12$ UDP connections. As for the traffic, we used CBR (Constant Bit Rate).

For the second scenario, we created 10 nodes in a 700m x 700m area and

established 8 UDP connections among the nodes. We used CBR for the traffic and Ad hoc On Demand Distance Vector (AODV) as the routing protocol in both of the scenarios.

We used the data rate of each UDP connection as the parameter for the search algorithms to tune. The data rate level may cause interference in the environment and thus affect the aggregate throughput achieved by the network. We used a lower bound of 0.01 Mb and an upper bound of 2 Mb for the data rates of traffic sources to tune. As the step size for search algorithms, we used 0.01Mb.

Figure 4.11 shows the results of experiments on 4 nodes and 12 UDP connections for different budgets with 80% confidence intervals. The horizontal axis represents number of samplings for each search algorithm. Because our sampling interval time is one second, the total budget also corresponds to simulation durations in seconds. The vertical axis represents the aggregate throughput in the network. We ran simulations for each duration 10 times and the values in the figure are the average of those 10 runs. The results are very promising. For 3 out of 5 cases, the PTAS found the best result among three search algorithms. For 2 out of 5 runs, the RRS [56] algorithm found the best result, and PTAS found the second best result. In general, PTAS seems to provide a quick and effective way of selecting the appropriate search algorithm for the problem at hand. In this particular case, RRS seems to perform the best among the three search algorithms, and PTAS picks RRS for most of the time and performs either better or very close to the performance of RRS.

Figure 4.12 shows the results of experiments on 10 nodes and 8 UDP connections for different runs. The horizontal axis represents run number of the simulations. These run numbers are used as the seed for the random number generator of the simulator. The vertical axis represents the aggregate throughput in the network. The

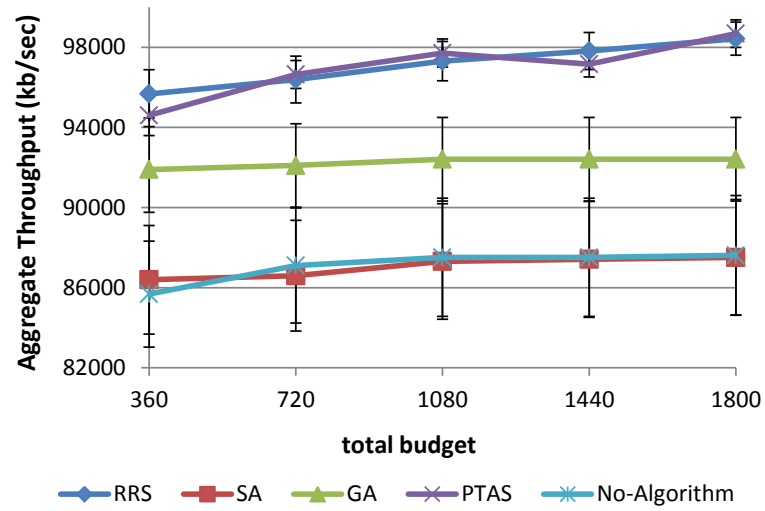


Figure 4.11: Results of experiments on 4 nodes and 12 UDP connections with different budgets.

simulation duration for these experiments is 720 second. The results are also promising in this graph. Because of its hybrid nature, PTAS finds either the best or the second best results in most of the cases.

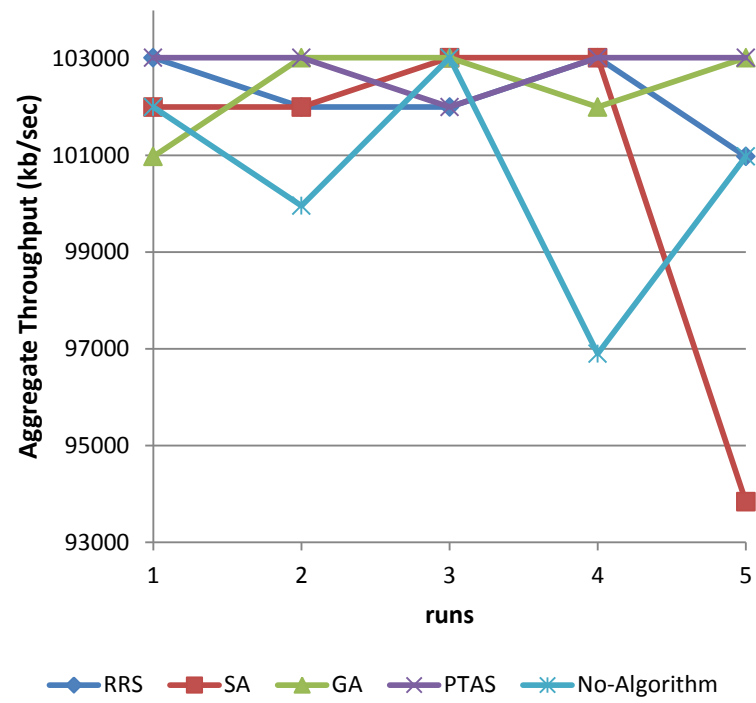


Figure 4.12: Results of experiments on 10 nodes and 8 UDP connections with different runs.

4.3 PTAS with No System Model

Online system optimization can be performed in two ways: (i) using a separate model of the system for experimenting new configurations, (ii) using the system itself for experimentation without a separate system model.

In the first version of PTAS [19], we took the former approach which is illustrated in Figure 4.1. The framework included two cycles: an optimization cycle and a deployment cycle. The deployment cycle takes place at a time scale where it is possible to configure a new set of parameters in the actual network. The optimization cycle iterates through a model of the system, for which we used a network simulator, and attempts to find the best possible set of parameters for the next deployment cycle. Hence, the frequency of the deployment cycle is smaller than the optimization cycle. The underlying assumption in this approach is that the behavior of the system-at-hand does not significantly change during at least one deployment cycle. This assumption is generally true for networks with a lot of fixed components such as backbone networks or wireless mesh networks. However, this approach fails when the network system is dynamic with high failure rates or a variable demand profile. For many practical networks, such as mobile ad-hoc networks (MANETs), the optimum set of parameters is no longer optimum after a short period. Thus, it is not practical to model such highly variant networks by simulations or other characterization techniques. To address the management of such highly dynamic networks, we take the model-free approach in this dissertation. We try to answer the key question of *“Is it possible and useful to use the actual network as the model and try out new parameter configurations on the real network with the hope of finding better ones?”*

As illustrated in Figure 4.2, the optimization and deployment cycles are merged.

We essentially perform in-situ trials of new configurations in the network itself to find better configurations. The key design issue is to search for better configurations without disturbing or temporarily deteriorating the network’s performance. The intuitive motivation is that the network’s performance may already be low since its dynamics are constantly changing. However, it is also possible the network’s performance to be noticeably disturbed as we search for better configurations.

Our approach employs two subsequent phases, *search* and *no-search*, where new configuration parameters are tried during the search phase. Each search phase is followed by a no-search phase with normal operation using the parameters found in the last search phase. In this design, the following questions state some of the key tradeoffs:

- *How frequent the search should be done?*
- *How long should the search phase be?*
- *How worse the search phase can temporarily make the system performance due to its trials?*

In the PTAS with no system model, we used an NS-2 simulator of a network to emulate a real-time running system and attempted to optimize the IGP link weights to obtain higher aggregate throughput. During the search phases, we used our black-box optimization algorithm, called Probabilistic Trans-Algorithmic Search (PTAS), to search for better IGP link weights. The PTAS framework can automatically determine the best set of algorithms that should work on the problem-at-hand. It is also adaptive to changes in the system behavior. This feature is especially useful for systems involving unexpected failures causing the response behavior to change, e.g., router or link failures in a network. We compared PTAS’ performance against

three other black-box optimization algorithms: Recursive Random Search (RRS) [56], Simulated Annealing (SA) [28], and Genetic Algorithm (GA) [18].

4.3.1 IGP Link Weights Optimization

To illustrate the plausibility and efficiency of our two-phase approach, we applied our framework on the setting of Interior Gateway Protocol (IGP) link weights.

We used NS-2 simulations of a sample IGP to mimic a physical network and evaluate the performance of the two-phase optimization using PTAS with no system model. We compared the performance of PTAS against the other three black-box optimization algorithms: Simulated Annealing (SA) [28], Genetic Algorithm (GA) [18], and Recursive Random Search (RRS) [56]. We used the Exodus topology from the Rocketfuel dataset. There are 22 nodes and 37 links in the Exodus topology. We identified 18 of the 22 nodes as edge nodes and established $18 \times 17 = 306$ TCP flows between them. The goal was to maximize the aggregate throughput in the network, which is calculated as the total number of bytes received at sink nodes of the TCP flows. In our experiments, we picked 10 links to fail during the simulations, and ran the experiments 10 times for each link to gain confidence with a different seed.

In our work with PTAS with separate system model [19], initially, we randomly assign IGP link weights and run the simulation for a particular period. This period is what we call as “total budget”, after which the metric to be optimized (e.g., the aggregate throughput) is sent to PTAS. In the next round, PTAS changes the link weights and runs the simulation again to obtain a new metric value based on the new link weights. By repeating this iterative process for a pre-defined amount of time (i.e., the deployment cycle in Figure 4.1), PTAS tries to determine the optimum set of link weights. This method is appropriate when the objective function does not

change during a complete deployment cycle.

In our work with PTAS with no system model, we do not run the network simulator multiple times because we treat the simulator as the physical in real time. We, again, define the optimization metric as the total number of bytes received at sink nodes of the TCP flows within a time interval. We call this time interval the “sampling interval” in our experiments. The number of metric samples to take is specified by the “total budget”. The simulator returns the metric, aggregate throughput, for the last sampling interval to the PTAS. Later, PTAS changes the link weights in the topology. As a result of these link weight changes, the routing and the paths taken by the TCP flows change, and thus the aggregate throughput changes. This framework establishes a relationship between the sampling interval, the number of samples (i.e., total budget), and the search phase duration:

$$search_phase_duration = sampling_interval * total_budget$$

At the end of a search phase, the two-phase system deploys the parameters which yield the optimum throughput. The system uses these link weights until the next search phase begins. We call the time between the search phases the “deployment phase”, or no-search phase, and it is related to the frequency of going into a search phase:

$$no_search_phase_duration = 1/search_frequency$$

Figures 4.13- 4.16 illustrates aggregate throughput versus time for each algorithm when they are applied within our two-phase optimization framework. The total simulation duration is 17,280 seconds, and the *no_search_phase_duration* is 5,000 sec-

onds. In order to simulate an unstable and dynamic environment, we randomly fail links with an exponentially distributed inter-failure time and failure duration. As can be observed, in some cases the throughputs after the search phases are worse than the throughputs before the search phases. The reason is that the environment is so unstable that link failures occur even during the search phases. We observed that PTAS handles the link failures well and improves the throughputs, particularly when the failures occur during the search phases. The average of the throughputs for each of the sampling intervals shown in Figures 4.13, 4.14, 4.15, and 4.16 are 7,698.24, 7,322.22, 7,596.68, and 7708.21 for RRS [56], SA [28], GA [18], and PTAS, respectively.

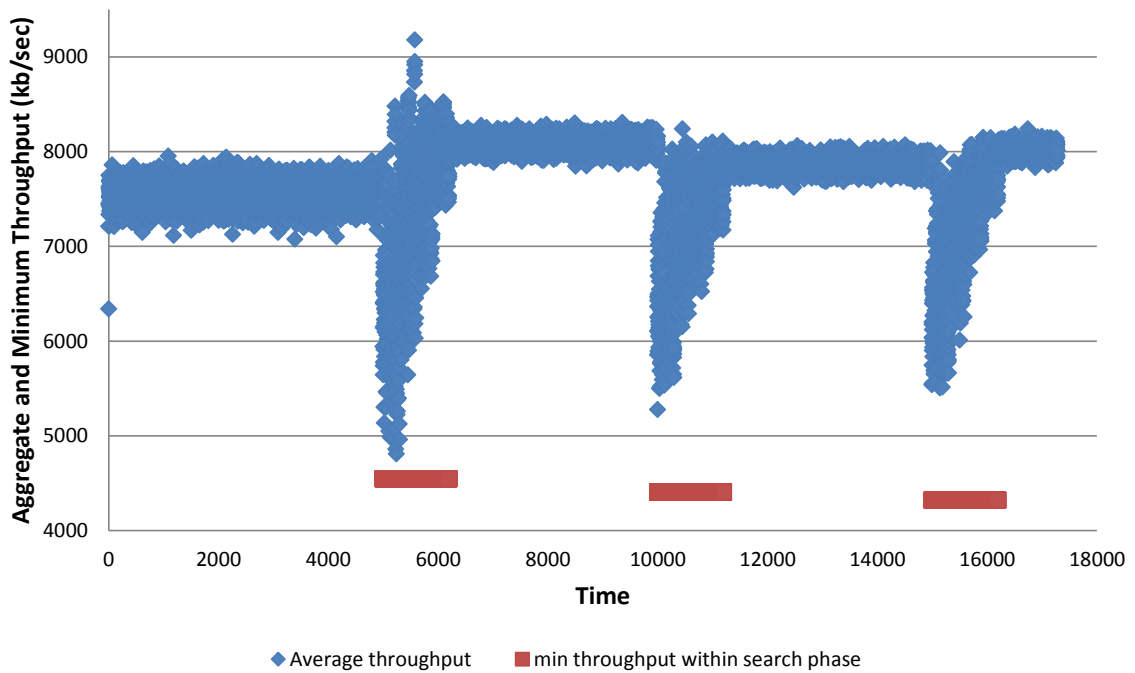


Figure 4.13: Optimization of a real-time simulator by using RRS.

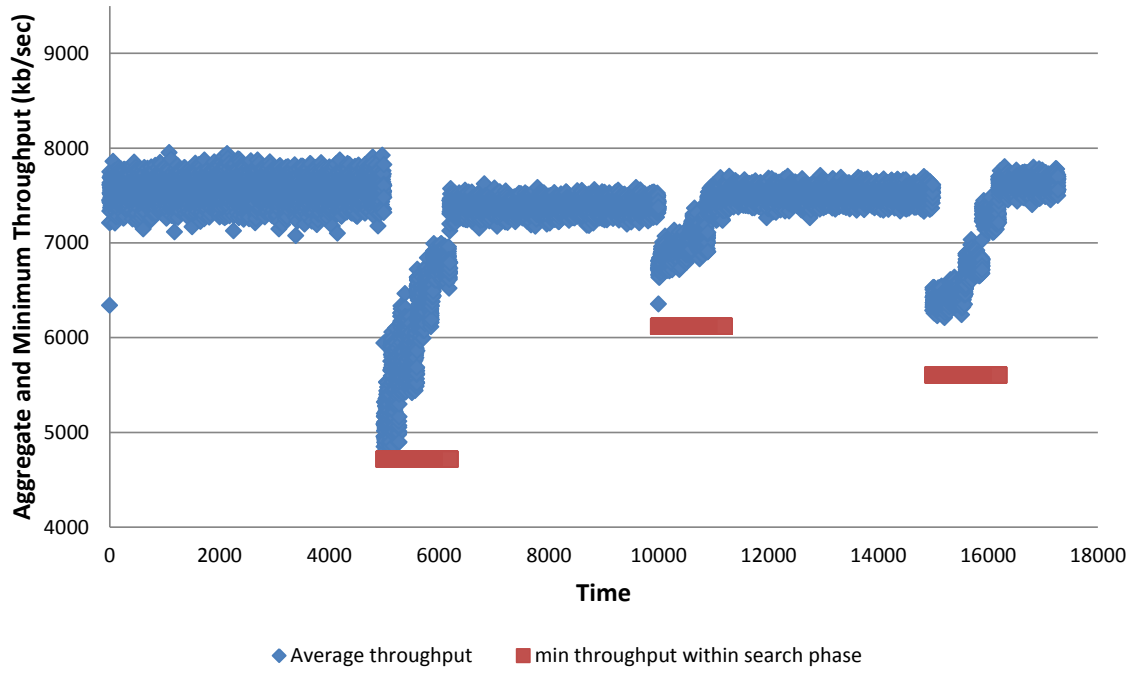


Figure 4.14: Optimization of a real-time simulator by using Simulated Annealing.

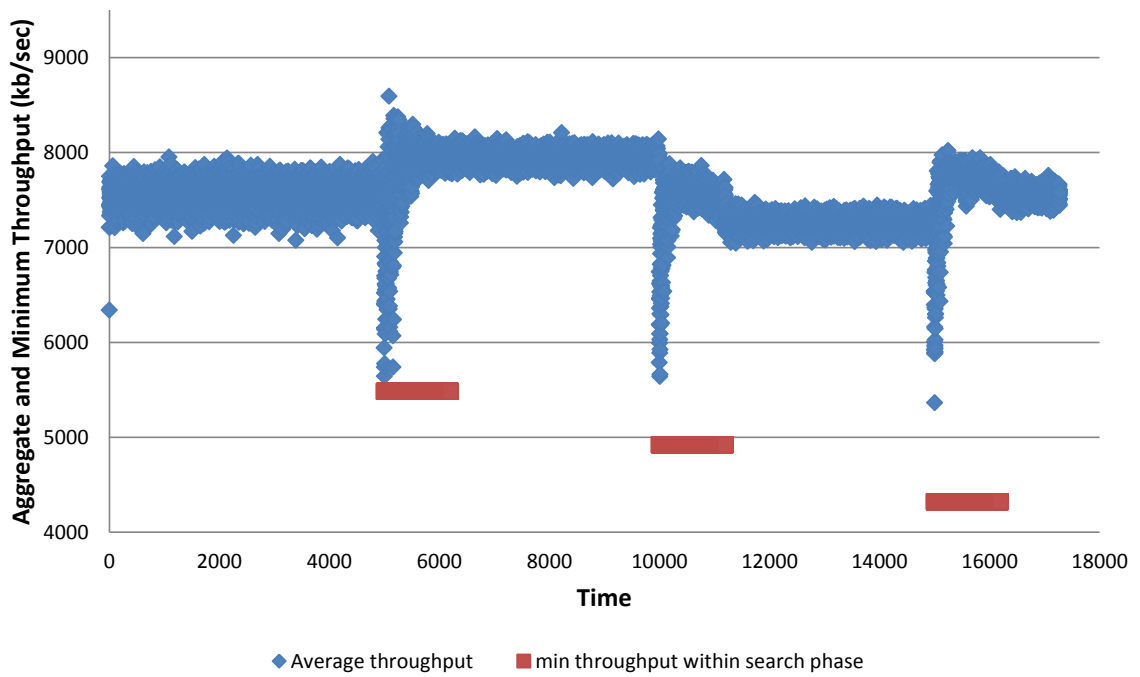


Figure 4.15: Optimization of a real-time simulator by using Genetic Algorithm.

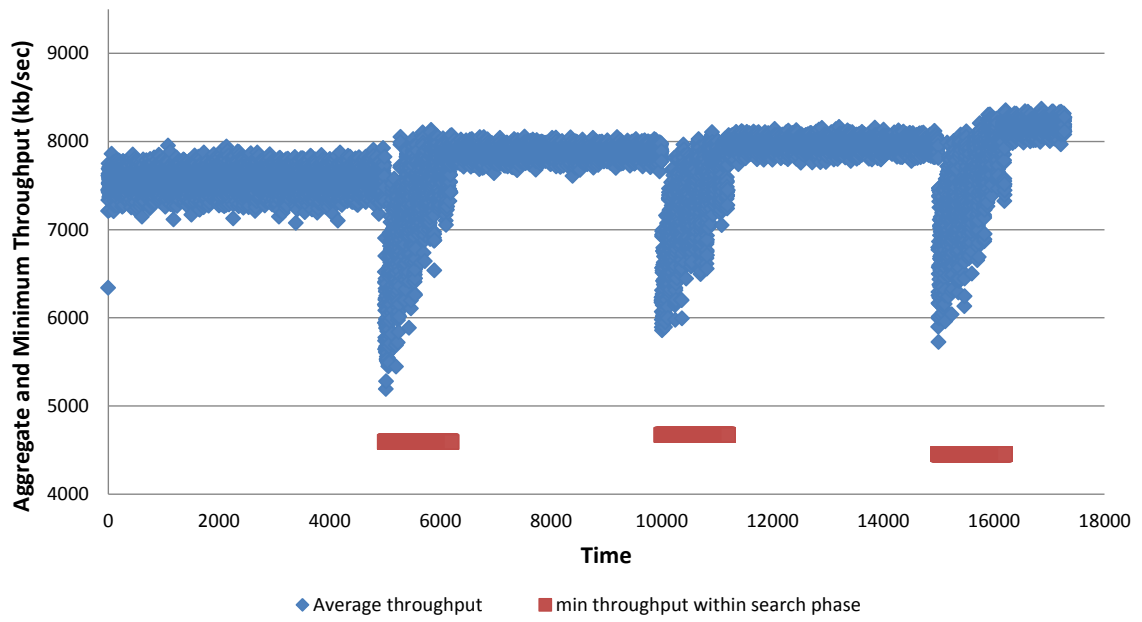


Figure 4.16: Optimization of a real-time simulator by using PTAS.

One important design tradeoff of our two-phase approach is to find an answer to the question: *How worse the search phase can temporarily make the system performance due to its trials?* The network's performance can be disturbed when we are searching for better configurations. Figures 4.13- 4.16 illustrate how worse the search phase can temporarily make the system performance due to its trials. PTAS and RRS achieve better throughput with slightly lower minimum throughput values in comparison to GA [18] and SA [28]. This shows that GA [18] and SA [28] are more exploitive algorithms. PTAS and RRS perform well in finding a good balance between exploitation (i.e. higher average throughput) and exploration (i.e. lower minimum throughput); and they do not bring the average throughput to unreasonably low values while trying to maximize it.

We also compared the performance of the PTAS with other algorithms on different link failure rates. Figure 4.17 shows the performance of the algorithms for various link failure frequencies. We used two different values for the *number_of_rounds* parameter of PTAS: 5 or 10. As expected, the aggregate throughput decreases when link failure frequency increases. Overall, PTAS performs similar to RRS [56] but noticeably better than GA [18] and SA [28]. We repeated the simulations 20 times with different seeds.

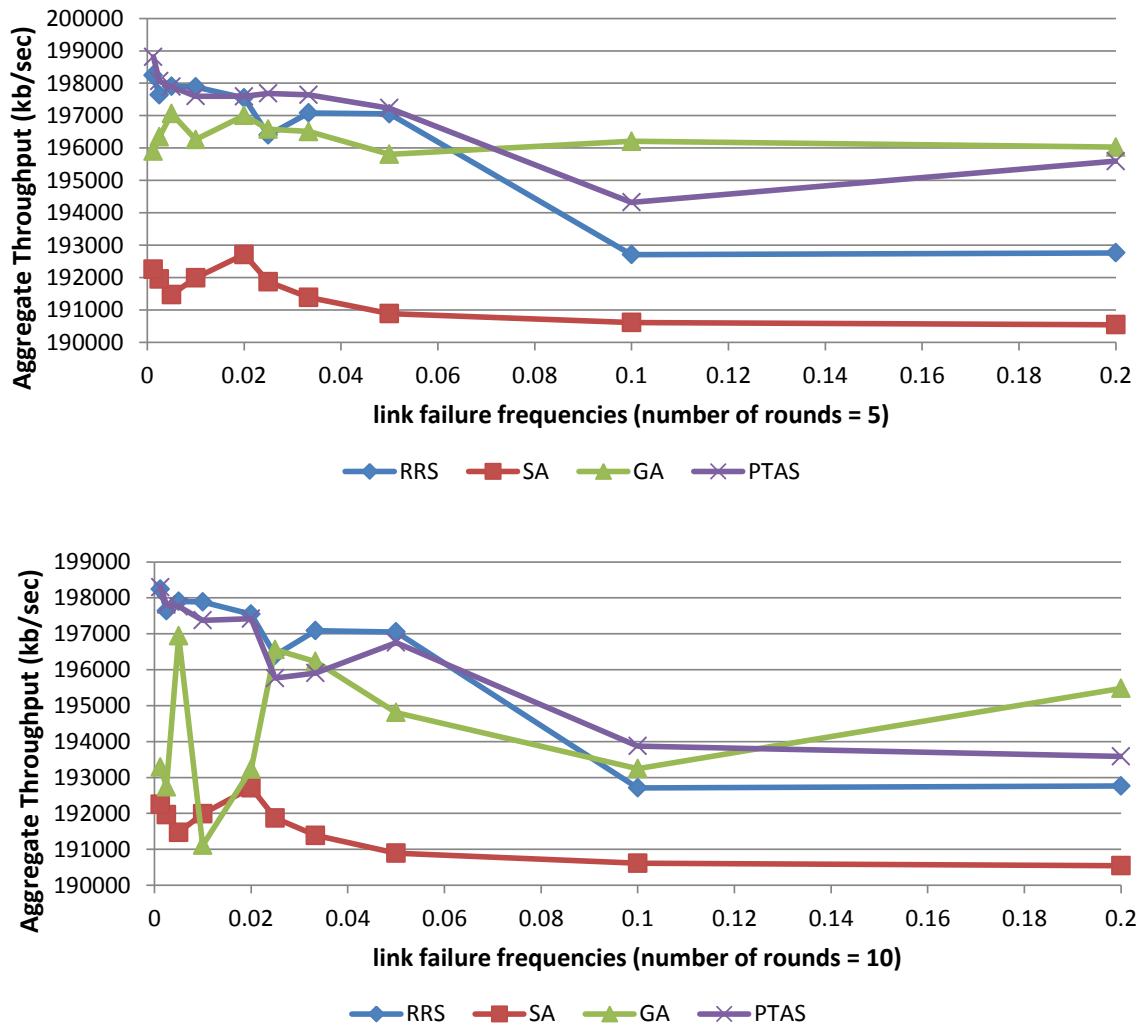


Figure 4.17: Comparison of PTAS with RRS, SA, and GA over different link failure frequencies.

A key tradeoff to be observed in the system is *how frequently the system should go into search phase*. We used five different values of *no_search_phase_duration*: 1000, 2000, 3000, 4000, and 5000. We observed that based on how dynamic the system is, there is an equilibrium point for PTAS. As we can see from Figure 4.18, going into the search phase every 3000 time units seems to achieve the best results for PTAS in this particular case. Going into search phase less frequently reduces the aggregate throughput because the system is too dynamic and the optimum set of parameters found in one search phase may not be optimum until the next search phase, and going into another search phase more frequently than 1000 is needed. On the other hand, going into the search phase more frequently than 3000 reduces the aggregate throughput because according to “No Free Lunch” theorem, there is a cost for the search phase. Because this is a black-box search, for some of the parameters used, the system may result in low metric output, reducing the aggregate throughput.

The other key tradeoff to be examined is *how long a search phase should last*. That is, what should the *total_budget* for a search phase be? We used four different *total_budget* values: 300, 600, 900, and 1200. We observed that some algorithms improve the overall average throughput when they are given longer search phase intervals, while other algorithms degrade. Because the problem is a black-box problem and that the PTAS is a hybrid search algorithm with better adaptivity, it tends to outperform the other three algorithms regardless of the total budget. Lastly, a key parameter for PTAS itself is what the number of rounds should be for PTAS. That is, when PTAS is given a total budget to use in a search phase, into how many chunks the PTAS should divide this budget? We used four different values: 4, 6, 8, and 10. We observed that because of the fast dynamics of the system, PTAS tends to perform better when the number of rounds increases. This is also illustrated in Figure 4.19.

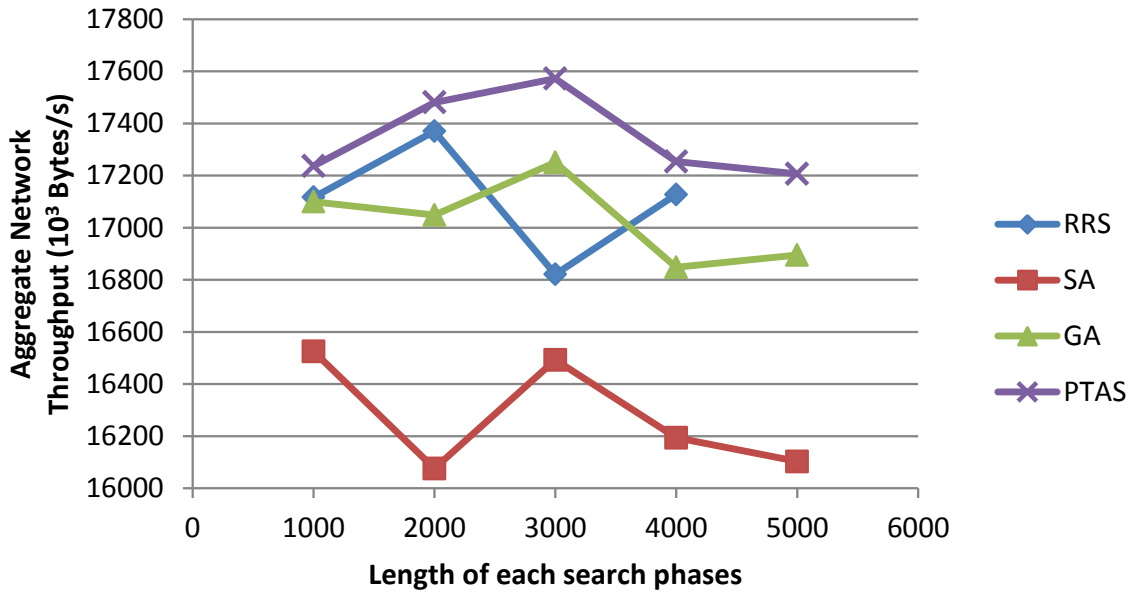


Figure 4.18: Comparison of PTAS with RRS, SA, and GA over different subsequent search phase intervals

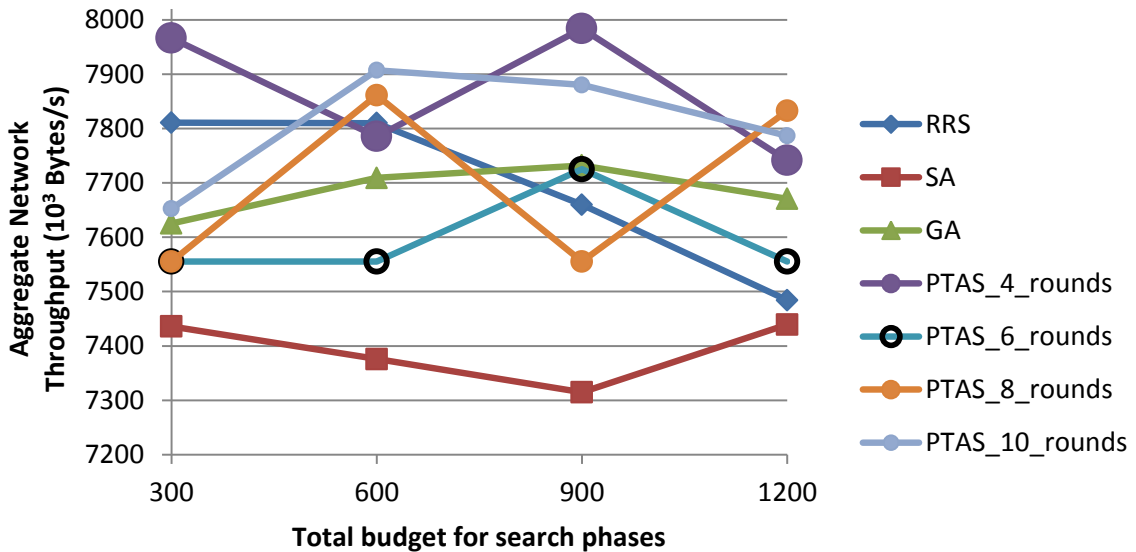


Figure 4.19: Comparison of PTAS with RRS, SA, and GA for using different search phase lengths and different number of rounds for PTAS

Chapter 5

Summary and Future Work

There are several algorithms tackling black-box problems, and some of these algorithms are hybrid algorithms comprised of multiple algorithms. Our PTAS framework presents a new approach in using a search algorithm to balance the experiment budget among multiple algorithms. We show how to hybridize three search algorithms with different characteristics such that some are good in exploration (Recursive Random Search [56] and Genetic Algorithm [18]) and others are good in exploitation (Simulated Annealing [28]).

We apply PTAS and three other search algorithms on six well-known objective functions, and run a number of experiments by using different prime numbers as the seed for random number generators. PTAS outperforms the other three algorithms on average.

Automated configuration and management of highly dynamic networks is a challenging problem for network practitioners. In this work, we used PTAS with no system model and PTAS with separate system model [19], on hybrid black-box optimization to adapt the search for changing network behavior. Using the system itself

for experimentation without a separate system model, we perform in-situ trials in a network for optimization. We investigate a two-phase online optimization framework for network configuration and management. Our approach follows a pattern of two subsequent phases, *search* and *no-search*, where new configuration parameters are tried during the search phase. We explore some of the key tradeoffs in the two-phase model, such as how frequent the search should be done, how long should the search phase be, and how worse the search phase can temporarily make the system performance due to its trials. We applied the two-phase optimization model on the IGP link weight setting problem and observe the aggregate network throughput against several parameters such as link failure rate, optimization algorithm, and total experiment budget the search phases have. For such dynamic scenarios, we showed that PTAS outperforms the individual algorithms by a sizeable margin on average. To illustrate applicability of our framework on wireless ad hoc network, we used NS-2 network simulator to optimize wireless ad hoc network by tuning data rates of traffic sources. The results are also promising. Because of its hybrid nature, the PTAS found either the best or the second best results in most of the cases.

We compare PTAS with the three search algorithms when the network system at-hand is very dynamic with factors, such as link failures and link recovery. PTAS performs better, and adapts to the new system more quickly due to its hybrid nature.

5.1 Future Work

We plan to apply PTAS to MANETs (Mobile Adhoc Networks) to optimize the throughput. We are planning to tune transmission data rates of traffic sources and transmission power for optimization.

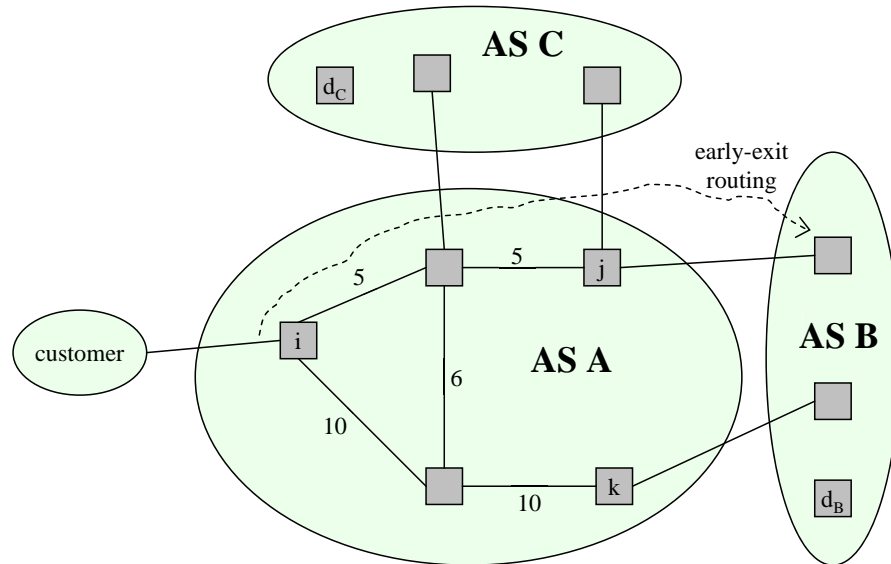


Figure 5.1: AS A with connections to AS B and AS C [45].

We would like to apply PTAS to the Border Gateway Protocol (BGP) link weights optimization problem. Internet traffic traverses several Autonomous Systems (ASes) from source to the destination. These Autonomous Systems may connect to each other at multiple locations. As we can see in the Figure 5.1, AS A let a customer reach some destinations in AS B and C via router i . To improve the reliability and performance, the ASes connect to each other at multiple locations. These ASes exchange reachability information by using a protocol, called the Border Gateway Protocol (BGP). Figure 5.1 shows that AS A and AS B connect to each other at two locations. For directing the traffic from router i to d_B , AS A has two possible egress points (routers j and k). In this case, BGP routing decision depends on the cost of the intradomain path to each egress router [45]. To avoid the congestions by load balancing, we plan to apply PTAS to this problem to optimize the performance of the network.

In this dissertation, we assume that the system-at-hand is black-box. However,

in some systems, we may have some information about the system. Importing such information into the PTAS will improve the optimization process. Being able to insert information about the system into the PTAS remains as a future work.

We would like to add more search algorithms into PTAS. That way, we can pick any number of search algorithms as we desire. We would like to analyze the mixture of the algorithms. Some questions to be answered; “What would be a good mixture for PTAS?”, “What would happen if we remove one particular search algorithm from the mixture?”

Bibliography

- [1] Geatbx: Example functions. <http://www.geatbx.com/docu/fcnindex-01.html>.
- [2] Ns2 network simulator. <http://www.isi.edu/nsnam/ns/>.
- [3] Rocketfuel: An isp topology mapping engine. <http://www.cs.washington.edu/research/networking/rocketfuel/>.
- [4] Simulated annealing. http://en.wikipedia.org/wiki/Simulated_annealing.
- [5] Throughput optimization for wireless data transmission. <http://eeweb.poly.edu/dgoodman/lavery.pdf>.
- [6] Tutorial: Exploration vs. exploitation. <http://www.indigosim.com/tutorials/exploration/t0s1.htm>.
- [7] The unr research grid. http://hpc.unr.edu/wiki/index.php/Main_Page.
- [8] O. F. K. A. Aron, FO and M. Odhiambo. Distributed topology control algorithm to conserve energy in heterogeneous wireless mesh networks. In *General science, engineering and technology Wireless technologies*, pages 530–536. World Academy of Science, Engineering and Technology, 2008.
- [9] T. Baeck, D. Fogel, Z. Michalewicz, and T. Back. *Evolutionary Computation 1: Basic Algorithms and Operators*. Inst. for Physics, 2000.
- [10] R. Bellman. *Dynamic Programming*. Dover Publications, 2003.
- [11] C. Boutilier, D. C. Parkes, T. Sandholm, and W. E. Walsh. Expressive banner ad auctions and model-based online optimization for clearing. In *Proceedings of the 23rd national conference on Artificial intelligence - Volume 1*, pages 30–37. AAAI Press, 2008.
- [12] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, 2001.

- [13] F. Dehne, J.-R. Sack, and M. Smid. *Algorithms and Data Structures*. Springer, 1999.
- [14] Y. Diao, F. Eskesen, S. Froehlich, J. L. H. A. Keller, J. L. Hellerstein, A. Keller, L. F. Spainhower, and M. Surendra. Generic on-line discovery of quantitative models for service level management. In *Proceedings of the 8th IFIP/IEEE International Symposium on Integrated Network Management*, pages 157–170. Kluwer Academic Publishers, 2003.
- [15] B. Feiring. *Linear Programming*. Sage Publications, 1986.
- [16] B. Fortz and M. Thorup. Increasing internet capacity using local search. *Comput. Optim. Appl.*, 29:13–48, October 2004.
- [17] J. C. M. G. Karthiga, J. Benitha Christinal. Performance analysis of various ad-hoc routing protocols in multicast environment. *IJCST*, 2:161–165, March 2011.
- [18] D. E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, 1989.
- [19] B. Gonen, M. Yuksel, and S. Louis. Probabilistic Trans-Algorithmic search for automated network management and configuration. In *IEEE International Workshop on Management of Emerging Networks and Services (IEEE MENS 2010)*, Miami, Florida, USA, 12 2010.
- [20] P. V. Hentenryck and R. Bent. *Online Stochastic Combinatorial Optimization*. The MIT Press, 2009.
- [21] A. Homaifar, A. Esterline, and B. Kimiaghalam. Hybrid projected gradient-evolutionary search algorithm for mixed integer nonlinear optimization problems. April 2005.
- [22] C.-T. Hsiao, G. Chahine, and N. Gumerov. Application of a hybrid genetic/powell algorithm and a boundary element method to electrical impedance tomography. *J. of Computational Phy.*, 173(2), 2001.
- [23] F. Hutter, H. H. Hoos, K. Leyton-Brown, and T. Stützle. Paramils: an automatic algorithm configuration framework. *J. Artif. Int. Res.*, 36:267–306, September 2009.
- [24] M. T. Jones. *AI Application Programming*. Charles River Media, 2003.
- [25] V. Kelner, F. Capitanescu, O. Lonard, and L. Wehenkel. A hybrid optimization technique coupling an evolutionary and a local search algorithm. *J. of Computational and Applied Mathematics*, 215(2):448–456, 2008.

- [26] B. I. Kumova. Dynamically adaptive partition-based data distribution management. *Proc. of Workshop on Principles of Advanced and Distributed Simulation*, pages 292–300, 2005.
- [27] J. Kurose and K. Ross. *Computer networking: a top-down approach featuring the Internet*. Addison-Wesley, 2000.
- [28] P. J. Laarhoven and E. H. Aarts. *Simulated Annealing: Theory and Applications*. Springer, 1987.
- [29] Z.-J. Lee and C.-Y. Lee. A hybrid search algorithm with heuristics for resource allocation problem. *Information Sciences*, 173(1-3):155–167, 2005.
- [30] Y. Li, L. Qiu, Y. Zhang, R. Mahajan, and E. Rozner. Predictable performance optimization for wireless networks. *SIGCOMM Comput. Commun. Rev.*, 38:413–426, August 2008.
- [31] B. H. Liu, Y. Gao, C. T. Chou, and S. Jha. An energy efficient select optimal neighbor protocol for wireless ad hoc networks. In *Proceedings of the 29th Annual IEEE International Conference on Local Computer Networks*, LCN '04, pages 626–633, Washington, DC, USA, 2004. IEEE Computer Society.
- [32] H. R. Liu, Y. F. Shen, Z. B. Zabinsky, C.-C. Liu, and S.-K. Joo. Social welfare maximization in transmission enhancement considering network congestion. *IEEE T. on Pow. Sys.*, 23(3):1105–1114, 2008.
- [33] X. Liu, L. Sha, Y. Diao, S. Froehlich, J. L. Hellerstein, and S. Parekh. Online response time optimization of apache web server. In *Proceedings of the 11th international conference on Quality of service*, IWQoS'03, pages 461–478, Berlin, Heidelberg, 2003. Springer-Verlag.
- [34] Z. Liu, M. S. Squillante, and J. L. Wolf. On maximizing service-level-agreement profits. In *Proceedings of the 3rd ACM conference on Electronic Commerce*, EC '01, pages 213–223, New York, NY, USA, 2001. ACM.
- [35] G. M. Lohman and S. S. Lightstone. Smart: Making db2 (more) autonomic. In *In VLDB 2002 28th International Conference on Very Large Data Bases*, Kowloon Shangri-La Hotel, Hong Kong, 2002.
- [36] D. A. Menascé, D. Barbará, and R. Dodge. Preserving qos of e-commerce sites through self-tuning: a performance model approach. In *Proceedings of the 3rd ACM conference on Electronic Commerce*, EC '01, pages 224–234, New York, NY, USA, 2001. ACM.

- [37] Z. Michalewicz and D. B. Fogel. *How to Solve It: Modern Heuristics*. Springer-Verlag, 2000.
- [38] R. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995.
- [39] J. A. Nelder and R. Mead. A simplex method for function minimization. *Computer Journal*, 7, 1965.
- [40] T. Niknam, B. B. Firouzi, and M. Nayeripour. An efficient hybrid evolutionary algorithm for cluster analysis. *World Applied Sciences Journal*, 4, 2008.
- [41] P. Pardalos, A. Migdalas, and R. E. Burkard. Combinatorial and global optimization. *Series on Applied Mathematics*, 14:55–74, 2002.
- [42] S. C. S. Porto and C. Ribeiro. A tabu search approach to task scheduling on heterogeneous processors under precedence constraints. *International Journal of High-Speed Computing*, 7, 1995.
- [43] M. Raghavachari, D. Reimer, and R. D. Johnson. The deployer’s problem: configuring application servers for performance and reliability. In *Proceedings of the 25th International Conference on Software Engineering, ICSE '03*, pages 484–489, Washington, DC, USA, 2003. IEEE Computer Society.
- [44] J. Rao, C. Zhang, N. Megiddo, and G. Lohman. Automating physical database design in a parallel database. In *Proceedings of the 2002 ACM SIGMOD international conference on Management of data, SIGMOD '02*, pages 558–569, New York, NY, USA, 2002. ACM.
- [45] J. Rexford. Route optimization in ip networks. In *in Handbook of Optimization in Telecommunications, Springer Science + Business*. Kluwer Academic Publishers, 2006.
- [46] A. Riedl. A hybrid genetic algorithm for routing optimization in ip networks utilizing bandwidth and delay metrics. In *Proc. of IEEE Workshop on IP Operations and Management*, pages 166–170, 2002.
- [47] N. Ruana and X. Sun. An exact algorithm for cost minimization in series reliability systems with multiple component choices. *Applied Mathematics and Computation*, 181(1):732–741, 2006.
- [48] M. Sniedovich. *Dynamic Programming*. CRC Press, 1992.
- [49] A. Sridharan and R. Gurin. Making igp routing robust to link failures. In *in IFIP-TC6 Networking Conference (Networking, 2005)*.

- [50] T. suk Kim. Improving spatial reuse through tuning transmit power, carrier sense threshold, and data rate in multihop wireless networks. In *In Proc. of ACM MobiCom*, pages 366–377. ACM Press, 2006.
- [51] J. A. Szab. An efficient hybrid optimization procedure of adaptive partition-based search and downhill simplex methods for calibrating water resources models. *Geophysical Research Abstracts*, 10, 2008.
- [52] R. Tronci, G. Giacinto, and F. Roli. Selection of experts for the design of multiple biometric systems. In *Proceedings of the 5th international conference on Machine Learning and Data Mining in Pattern Recognition, MLDM '07*, pages 795–809, Berlin, Heidelberg, 2007. Springer-Verlag.
- [53] P. J. Vaccaro. The 80/20 rule of time management. *Family Practice Management*, 7:76, September 2000.
- [54] V. V. Vazirani. *Approximation Algorithms*. Springer Verlag, 2002.
- [55] D. Wang, G. Li, and R. Doverspike. Igp weight setting in multimedia ip networks. In *INFOCOM 2007. 26th IEEE International Conference on Computer Communications. IEEE*, pages 2566 –2570, may 2007.
- [56] T. Ye and S. Kalyanaraman. A recursive random search algorithm for black-box optimization. *ACM SIGMETRICS Performance Evaluation Review*, 32(3):44–53, December 2004.
- [57] T. Ye, H. T. Kaur, S. Kalyanaraman, K. S. Vastola, and S. Yadav. Dynamic optimization of ospf weights using online simulation, 2002.
- [58] T. Ye, H. T. Kaur, S. Kalyanaraman, and M. Yuksel. Large-scale network parameter configuration using an on-line simulation framework. *IEEE/ACM Tran. on Networking*, 16(4):777–790, 2008.
- [59] J. Yen, D. Randolph, J. C. Liao, and B. Lee. A hybrid approach to modeling metabolic systems using genetic algorithm and simplex method. *P. of Conf. on AI for Apps.*, 20-23:277–283, 1995.
- [60] M. Zdansky and J. Pozivil. Combination genetic/tabu search algorithm for hybrid flowshops optimization. *P. of ALGORITMY*, pages 230–236, 2002.
- [61] A. A. Zhigljavsky and J. D. Pinter. *Theory of Global Random Search*. Springer, 1991.