A Unified Search Framework for Large-scale Black-box Optimization

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

ABSTRACT

The parameter configuration of a network protocol can be formulated as a black-box optimization problem with network simulation evaluating the performance of the black-box, i.e., the network. This paper proposes a unified search framework (USF) to handle such large-scale black-box optimization problems. The framework is designed to provides a general platform on which tailored optimization algorithms can be constructed easily for various types of problems. Therefore, it can be applied to the configuration of different network protocols. In the USF, various samplers are provided as basic building blocks and each of them implements a certain search technique. For a specific problem, a selection of samplers can be used to construct an appropriate search algorithm. These samplers are run in parallel and coordinated with various type of memories which selectively store the samples generated by samplers. The USF also include a resource management mechanism, which can manage parallel computing devices, for example, a network of workstations, and allocate the available computing resources to samplers according to the predefined allocation strategy. The benchmark tests are presented in this paper to demonstrate the flexibility and advantages of the USF.

1. INTRODUCTION

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

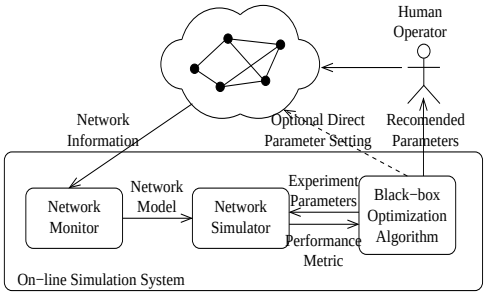

Figure 1: On-line simulation framework for network protocol optimization

Optimization problems often arises in practice and they can be formulated as (assume minimization): given a real-valued objective function $f : \mathbb{R}^n \rightarrow \mathbb{R}$, find a global minimum,

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

where D is the parameter space, usually a compact set in $\mathbb{R}^n$. The objective function in practice is often non-linear and may have multiple local optima. Furthermore, *a priori* knowledge about the objective function is usually very limited and such problems are termed as "black-box" optimization. That is, the mathematical formula of the objective function is unknown and the function value can only be evaluated with computer simulation or other indirect ways.

A large number of optimization algorithms have been proposed to solve black-box optimization problems, such as, controlled random search[2], genetic algorithm[3] and simulated annealing[4]. These algorithms have been demonstrated to be successful in many applications to practical problems. However, no analytical conclusion has been made on which optimization algorithm is the most efficient. Furthermore, there has been no consistent empirical report in this aspect. No Free Lunch (NFL) Theorem[5] has mathematically proved that the average performance of any optimization algorithm is the same over all possible problems. Note that NFL only implies that there is no general applicable optimization algorithm. However, for a specific class

of problems, an optimization algorithm may perform more efficiently than another if it performs better in exploiting the intrinsic structural properties of the underlying problem. Since one single optimization technique is often insufficient to handle complex practical problems, different search techniques have been combined in practice for better performance, for example, combining local search techniques with GA, simulated annealing, or controlled random search[6, 7, 8], or combining GA with simulated annealing[9]. In fact, any optimization algorithm is just a combination of simple search techniques. For example, multistart local search is composed of random sampling and local search. The task of an optimization algorithm is just to coordinate these techniques to achieve the best performance.

With the increasing availability of parallel computing devices, such as, parallel computers or computer farms composed of a network of workstations, the general solution for practical optimization problems also needs a resource manage mechanism to fully utilize these resources. This is very important for practical optimization problems where the objective is to make the most use of a certain supply of computing resources for the maximum efficiency. In fact, the existence of optimization algorithms is due to the limit of available computing resources. If we had an infinite supply of computing resources which can evaluate any number of samples at one time, there would be no need for optimization algorithms. A brutal enumeration method will find the exact global optimum. Therefore, from a practical point of view, the task of an optimization algorithm is just to allocate the limited computing resources among the underlying search techniques appropriately such that the desired efficiency can be achieved.

Based on the above ideas, this paper proposes a Unified Search Framework as a general solution to black-box optimization problems. Basically, the USF includes various simple search techniques, such as random sampling and pattern search, as basic building blocks. For a practical problem, a selection of building blocks are combined and run *in parallel*. USF includes a resource allocation and management mechanism to allocate computing resource among the search techniques and fully utilize available computing resource. The resource management of USF is based on the assumption that function evaluation consumes most of resource in optimization, and compared with this, the consumption of other operations can be ignored. The resource management of USF is scalable in heterogeneous computing environment while traditional parallel optimization algorithms are designed to exploit a specific parallel computing infrastructure.

The rest of the paper is organized as follows: Section2 describes the Unified Search Framework and its major components. Section3 presents the benchmark tests and demonstrates the flexibility and advantages provided by USF. Finally, Section4 concludes this paper and discuss further research directions.

2. THE UNIFIED SEARCH FRAMEWORK

The basic structure of Unified Search Framework is shown in Figure 2. Two basic components in USF are: *sampler* and *memory*. In USF, any algorithm is established upon these two type of components. A sampler implements a certain search technique, such as, random sampling and pattern search, and a memory is used to store sampling points

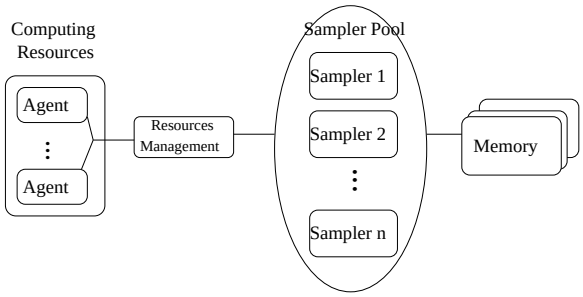

Figure 2: The Unified Search Framework

which are generated by a sampler and may be used in the future optimization process. For one specific problem, we can choose a few types of samplers and combine them together with the help of various types of memory for the most efficient optimization. Another important component of USF is its computing resource management. In one USF type optimization algorithm, the samplers are run in parallel with assigned allocations of computing resources. The computing resource management will maintain the resource usage of each samplers to its allocation.

Samplers in USF implement various search technique. For a specific problem, appropriate samplers should be chosen and combined based on the features of the problem. It is very important to include an proper selection of samplers which best exploit the structures of the underlying objective function. To achieve this, the underlying features of the objective function have to be carefully investigated.

Memory selectively stores the sampling points output by samplers and these samples can be used by other samplers in later optimization process. The concept of memory is first proposed in tabu search algorithm[10], where memory is used to prevent the revisit of the sampling points already examined. Basically, any optimization algorithm can take advantage of memory to improve the efficiency. Memory plays a more important role in USF than in tabu search. In USF, samplers are run independently in parallel. To coordinately with each other, they have to be coupled with various types of memories.

One example algorithm is shown in Figure 3. As shown in the figure, the algorithm is composed of two samplers: random sampling and pattern search. The samples generated by random sampling are put into a "drop-head" memory, which drops the oldest samples when the memory reaches its capacity. Pattern search uses these samples as the starting points to perform local search. Therefore, it basically implements a multi-start pattern search algorithm. In this algorithm, random sampling and pattern search are executed independently in parallel and their cooperation is achieved by the help of drop-head memory. Besides the drop-head memory used in the example, many other types of memory can be used. For example, we can use a memory which stores only the best few samples or the samples which satisfy a certain criterion, say, better than a certain threshold. The choice of of memories may affect the optimization efficiency substantially and should be carefully made based on the features of the underlying problem.

With samplers and memories as building blocks, most of traditional optimization algorithm can be composed in the

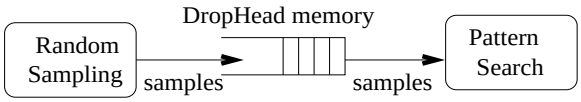

Figure 3: An example of samplers coupled with memory

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

2.1 Computing Resources Model and Management Mechanism

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

The resources management of USF is illustrated in Figure 4. The computing resources can be composed of multi-

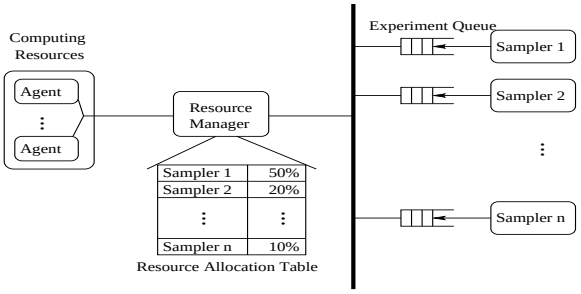

Figure 4: Resource allocation mechanism

processor parallel computers or a network of workstations. On each computing device, an agent is run to manage the device and communicate with the resource manager of USF. Whenever an agent finds that the managed computing device is free, it requests an experiment, i.e., function evaluation, from the manager. Each sampler maintains an experiment queue. Whenever the sampler generates a sample, it puts it into the queue for function evaluation. When the queue is full, the sampler will stop generating new samples until the queue becomes available again. When receiving a request from an agent, the resource manager will then examine the experiment queues of all running samplers and choose one to return according to the resource allocation rule.

To allocate computing resources among the samplers, the manager maintains a resource allocation table which defines the percentage of computing resources that each sampler can use. For example, Figure 5 shows a resource allocation table among 3 samplers: This table indicates that Sampler 1 may

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

Figure 5: An example resource allocation table

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

The manager also maintains another table which records the actual resource usage for each sampler and is initialized to be the same as the allocation table in the beginning. Whenever the manager receives a experiment request from computing resources, it will check those samplers with non-empty experiment queues, and choose the one with the maximum unused allocation in the usage table. After sending the selected experiment to computing resources, the manager updates the usage table by subtracting the chosen item by 1 and then adding each item in the usage table (including those with empty queues) by its corresponding allocation percentage. One example of this procedure is shown as in Figure 6. As shown in the example, the usage table is first initialized as the allocation table. Suppose the experiment queues of all samplers are not empty, sampler 1 is first chosen to send its experiment for evaluation since it has the maximum allocation. Then the usage of sampler 1 is subtracted by 1 as shown in table (b) and the usage of each sampler is added by its allocation as shown in table (c). After this, sampler 3 will be chosen next time since now it has the maximum value in the usage table.

By using the above method, the computing resources can be distributed among the samplers in accordance with the resources allocation table. The resource allocation table provides us with the flexibility to adjust the coordination between samplers running in parallel and achieve the best efficiency. In fact, many sequential optimization algorithms use certain mechanisms to adjust the computing resource allocation between search techniques. For example, in simulated annealing[4], the "temperature" parameter is used to control the balance between random walk and hillclimbing. With the high temperature in the beginning, random walk runs more frequently and hence uses more resources. With the temperature cooling down, hillclimbing gets more and more computing resources. In genetic algorithm, similar control parameters also exists, such as, crossover rate or

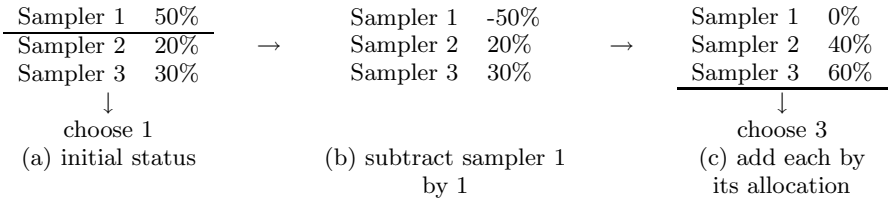

Figure 6: An example of resource allocation operation

mutation rate.

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

2.2 Parallel Optimization Strategy

One design objective of USF is to fully utilize available computing resources. To achieve this, All samplers in USF are run in parallel and their function evaluations are distributed to computing resources based on their allocation. As mentioned before, some search technique, especially local search methods, such as, hillclimbing, may not be able to fully use its allocated resources. For these techniques, a certain parallel optimization strategy has to be used for full resource utilization. Many parallel optimization algorithms has been proposed[11, 12]. The parallel strategies used in these algorithm can be classified into the following three categories:

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

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

Algorithm parallelization is to parallelize a sequential algorithm by careful changing its design. This method is specific to a certain search technique and hence is not generally applicable to other techniques. Furthermore, its parallelity is normally constrained by the inherent limit of the algorithm and not scalable with available computing resources.

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

3. TESTS OF UNIFIED SEARCH FRAMEWORK

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

3.1 Effect of Memory Types

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

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

Two samplers are used in the tests: random sampling and pattern search and they are coupled together with a certain memory just like the example shown in Figure 3. The optimization algorithms obtained with the above memories are tested on a benchmark function, i.e., 20-dimensional Rastrigin function. The convergence curves of two memory types are shown in Figure 7. We can see that by using drop-worst

memory, the optimization efficiency can be improved substantially.

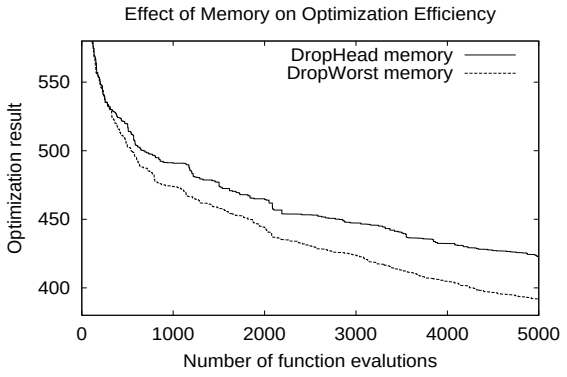

Figure 7: Effect of memory on optimization efficiency

3.2 Effect of Resource Allocation Strategy

In addition to memory, USF also provide a flexible resource allocation mechanism to adjust the coordination of samplers. In this section, we will examine the effect of resource allocation strategy on the optimization efficiency. We still use the same two samplers as before: random sampling and pattern search, however, we vary the resource allocation between these two samplers in the tests. The results are shown in Figure 8. The horizontal axis indicate the resource allocation of random sampling. Given the allocation of random sampling, pattern search will take all the remaining resources with multi-path strategy described before. For example, if random sampling gets 20% resources, pattern search then gets 80%. The vertical axis indicates the optimization result after 5000 function evaluations. As we can see from the figure, the optimization efficiency varies greatly with the resource allocations for the samplers. At one extreme, allocating all computing resources to random sampling is equivalent to a pure random sampling algorithm. At the other extreme, allocation most of resources to pattern search is equivalent to a simple multi-start pattern search algorithm. Neither case can produce good efficiency in the test. The best balance between two samplers is achieved at a point between two extreme case. As shown in the figure, allocating 60% resources to random sampling produces the best efficiency.

3.3 Scalability of Parallel Optimizaiton

One important design objective of USF is to fully utilize available computing resources. In this section, we examine if the optimization efficiency can be improved by making full use of computing resources. In the tests, we use the algorithm described in Figure 3 with a network of workstations to optimize 20-dimensional Rastrigin function. We vary the number of workstations in the tests and compare the optimization performance of USF with different number of workstations. To simulate the situation where function evaluations are expensive, we did not code the benchmark function directly into the optimization algorithm. Instead we used a slow script language to evaluate the benchmark function. The left plot in Figure 9 shows the optimization

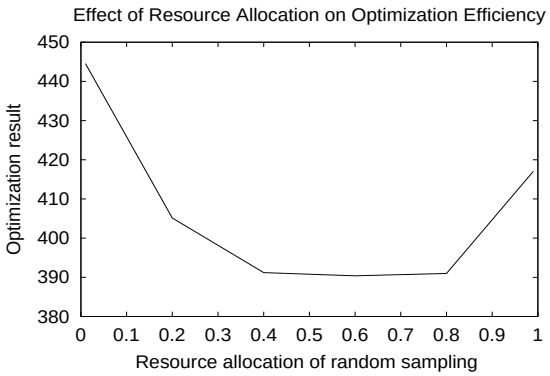

Figure 8: Effect of resource allocation on optimization efficiency

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

4. CONCLUSION

This paper presented a Unified Search Framework(USF) as a general solution for large-scale black-box optimization problems. Specifically, it can be used to optimize the configuration of various network protocols in an on-line simulation system[1]. In the USF, an optimization algorithm is considered to be an appropriate combination of search techniques and the optimization algorithm achieves the desired efficiency by properly coordinating these techniques and allocating the limited available computing resources among them. The USF includes various types of search techniques as the building blocks and use memories to couple these techniques together and hence obtain an optimization algorithm tailored for a specific problem. In addition, the USF comprises a scalable resource management mechanism to allocate available computing resource among search techniques and take full advantage of these resources.

The Unified Search Framework provides a flexible platform to construct optimization algorithms for various practical optimization problems. To achieve the best efficiency, appropriate search techniques should be first chosen. In addition, the coordination among these techniques should also be carefully adjusted based on the features of the underlying problem. This can be achieved in USF by adjusting resource allocation of these techniques and selecting memory types coupling these techniques. To make the correct selection and adjustment, the features of the problem have to be

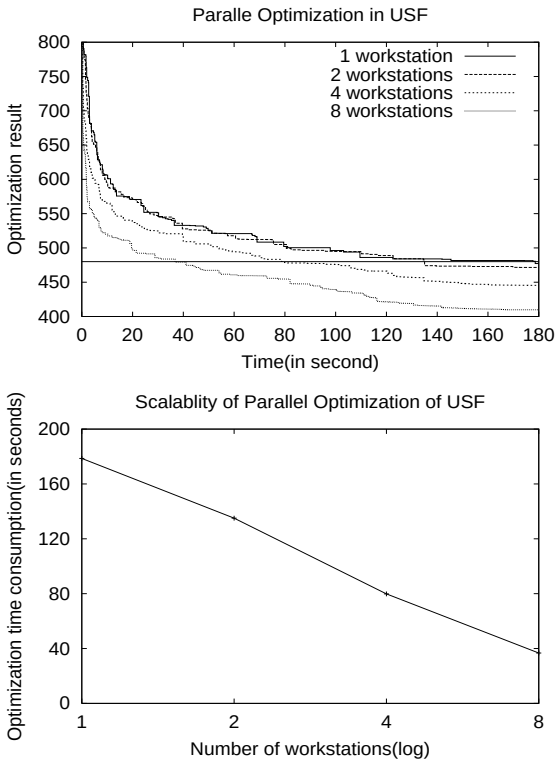

Figure 9: Scalability of parallel optimization in USF

carefully examined. Currently, this step has to be performed manually based on the practitioner's experience or trial and error. To automate this process, the correspondence between search techniques and suitable structures has to be established first. Basically, we need to know how to identify these structures and how they affect the coordination of search techniques. These issues are still open problems and will be investigated in our future work.

5. REFERENCES

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

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

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

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

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

[6] Ko-Hsin Liang, Xin Yao, and Charles Newton. Combining landscape approximation and local search in global optimization. In *Proc. of the 1999 Congress on Evolutionary Computation*, volume 2, pages 1514–1520, Piscataway, NJ, July 1999. IEEE Press.

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

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

[9] X. Yao. Optimization by genetic annealing. In M. Jabri, editor, *Proc. of Second Australian Conf. on Neural Networks*, pages 94–97, Sydney, Australia, 1991.

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

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

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