Inproc. of Eleventh International Conference on Systems Engineering, Las Vegas, NV, July 9 - 11, 1996

An Empirical Comparison of Seven Iterative and Evolutionary Heuristics for Static Function Optimization (Extended Abstract)

Shumeet Baluja

School of Computer Science, Carnegie Mellon University, Pittsburgh, Pennsylvania 15213.

Abstract

This report is a summary of the results obtained from a large scale empirical comparison of seven iterative and evolution-based optimization heuristics. Twenty-seven static optimization problems, spanning six sets of problem classes which are commonly explored in genetic algorithm literature, are examined. The search spaces in these problems range from 2^{300} to 2^{2040}. The results indicate that using standard genetic algorithms for the optimization of static functions does not yield a benefit, in terms of the final answer obtained, over simpler optimization heuristics.

1. Introduction

Genetic algorithms (GAs) and other evolutionary procedures are commonly used for static function optimization. Despite growing evidence that methods such as GAs are, in general, not well suited in this domain [De Jong, 1993], a large amount of research has been devoted to improving their effectiveness for function optimization. Hybrid mechanisms, such as specialized operators and representations which can intelligently use problem specific information, have achieved good results in specific applications. Nonetheless, relatively few of these techniques work well across a wide range of problems.

This study aims at addressing only one question: "How effective are standard GAs for optimizing static functions, given a set number of function evaluations, in comparison to other, simpler, algorithms?" The algorithms to which GAs are compared are multiple-restart stochastic hillclimbing (MRSH) and population-based incremental learning (PBIL). A total of three variants of MRSH, two variants of PBIL, and two GAs are compared. These optimization heuristics are compared on problems which are representative of those commonly attempted in GA literature.

It is important to understand the scope of these results. All of the empirical comparisons are based upon static function optimization problems. The performance of each method is judged solely by the best solution found during the run, given a pre-specified number of total evaluations. In an attempt to minimize the effects of hand-coding/tuning of the algorithms and problem representations, the algorithms are used with as little problem-specific knowledge as possible. The only problem-specific knowledge used in these algorithms is the number of bits in the solution encoding for each of the problems.

2. Algorithms Compared

The parameters for all of the algorithms were chosen to work well on many of the problems, but are not biased to any single problem. Additionally, the GAs were selected to perform well on the task of opti-

mization: they use mechanisms such as elitist selection and scaling of fitness values (described below), which are often useful for the optimization of static functions [De Jong, 1993].

2.1 Genetic Algorithms

In the standard GA, candidate solutions are encoded as fixed length binary vectors. The initial group of potential solutions is chosen randomly. These candidate solutions, called "chromosomes," evolve over a number of generations. At each generation, the fitness of each chromosome is calculated; this is a measure of how well the chromosome optimizes the objective function. The subsequent generation is created through a process of selection, recombination, and mutation. The chromosomes are probabilistically selected for recombination based upon their fitness. General recombination (crossover) operators merge the information contained within pairs of selected "parents" by placing random subsets of the information from both parents into their respective positions in a member of the subsequent generation. Although the chromosomes with high fitness values have a higher probability of selection for recombination than those with low fitness values, they are not guaranteed to appear in the next generation. Due to the random factors involved in producing "children" chromosomes, the children may, or may not, have higher fitness values than their parents. Nevertheless, because of the selective pressure applied through a number of generations, the overall trend is towards higher fitness chromosomes. Mutations are used to help preserve diversity in the population by introducing random changes into the chromosomes. Detailed discussions of GAs can be found in [Goldberg, 1989] [De Jong, 1975] [Holland, 1975].

Two variants of the standard GA are tested in this study. The first, SGA, has the following parameters: Two-Point crossover, with a crossover rate of 100%, mutation rate of 0.001, population size of 100, and elitist selection (the best chromosome in generation N replaces the worst chromosome in generation N+1). The second GA used, termed GA-Scale, uses the same parameters with the following exceptions: uniform crossover with a crossover rate of 80%, and the fitness of the worst member in a generation is subtracted from the fitnesses of each member of the generation before the probabilities of selection are determined. Both GAs are generational, and both employ the elitist selection mechanism described above.

2.2 Multiple-Restart Stochastic Hillclimbing

Three variants of Multiple-Restart Stochastic Hillclimbing (MRSH) are explored in this paper. In each of these variants, the initial point is chosen randomly, see Figure 1. The first version, MRSH-1, maintains a list of the position of the bit flips which were attempted without improvement. These bit flips are not attempted again until a better solution is found. When a better solution is found, the list is emptied. If the list becomes as large as the solution encoding, MRSH-1 is restarted at a random location with an empty list. MRSH-2 and MRSH-3 allow moves to regions of higher and equal evaluation. This is different than MRSH-1, which only allows moves to regions of higher evaluation. In MRSH-2, the number of evaluations before restart depends upon the length of the encoded solution. MRSH-2 allows 10*(length of solution) evaluations without improvement before search is restarted. When a solution with a higher evaluation is found, the count is reset. In MRSH-3, after the total number of iterations is specified, restart is forced 5 times during search, at equally spaced intervals.

2.3 Population-Based Incremental Learning

Population-based incremental learning (PBIL) is a combination of evolutionary optimization and artificial neural network learning [Baluja, 1994]. The object of the algorithm is to create a real valued probability vector which, when sampled, reveals high quality solution vectors with high probability. For

```
V ← randomly generate solution vector
Best ← evaluate (V)

loop # ITERATIONS
        N ← Flip_Random_Bit (V)
        if (evaluate (N) > Best)
                Best ← evaluate(N)
                V ← N
```

Flip_Random_Bit is a function which returns a solution string with only one bit changed from its input solution string.

Figure 1: General MRSH for binary solution vectors. The best vector and its evaluation can be saved. In practice, the algorithm can be restarted in random locations, and the best solution ***ever*** found returned.

```
****** Initialize Probability Vector ******
for i :=1 to LENGTH do P[i] = 0.5;

while (NOT termination condition)
        ****** Generate Samples ******
        for i :=1 to SAMPLES do
                sample_vectors[i] := generate_sample_vector_according_to_probabilities (P);
                evaluations[i] := evaluate(sample_vectors[i]);

        best_vector := find_vector_with_best_evaluation (sample_vectors, evaluations);
        worst_vector := find_vector_with_worst_evaluation (sample_vectors, evaluations);

        ****** Update Probability Vector Towards Best Solution ******
        for i :=1 to LENGTH do
                P[i] := P[i] * (1.0 - LR) + best_vector[i] * (LR);


        ****** Update Probability Vector Away from Worst Solution ******
        for i :=1 to LENGTH do
                if (best_vector[i] ≠ worst_vector[i]) then
                        P[i] := P[i] * (1.0 - NEGATIVE_LR) + best_vector[i] * (NEGATIVE_LR);


        ****** Mutate Probability Vector ******
        for i :=1 to LENGTH do
                if (random (0,1) < MUT_PROBABILITY) then
                        if (random (0,1) > 0.5) then mutate_direction := 1
                        else mutate_direction := 0;
                        P[i] := P[i] * (1.0 - MUT_SHIFT) + mutate_direction * (MUT_SHIFT);
```

PBIL: USER DEFINED CONSTANTS (Values Used in this Study):
SAMPLES: the number of vectors generated before update of the probability vector (100).
LR: the learning rate, how fast to exploit the search performed (0.1).
NEGATIVE_LR: negative learning rate, how much to learn from negative examples (PBIL1=0.0, PBIL2= 0.075).
LENGTH: the number of bits in a generated vector (problem specific).
MUT_PROBABILITY: the probability for a mutation occurring in each position (0.02).
MUT_SHIFT: the amount a mutation alters the value in the bit position (0.05).

Figure 2: The PBIL1/PBIL2 algorithm for a binary alphabet. Only PBIL2 includes shaded region.

example, if a good solution can be encoded as a string of alternating 0's and 1's, a suitable final probability vector would be 0.01, 0.99, 0.01, 0.99, etc. The exact algorithm and parameters are shown in Figure 2. The relationship between PBIL and GAs is described in [Baluja and Caruana, 1995], see also [Juels, 1996].

3. A LARGE-SCALE EMPIRICAL COMPARISON

In this section, the algorithms described previously are applied to six classes of problems: Traveling Salesman, jobshop scheduling, knapsack, bin packing (cutting stock), neural network weight optimization, and numerical function optimization. The results obtained in this study should *not* be considered to be state-of-the-art. The problem encodings were chosen to be easily reproducible, and to allow easy and fair comparison with other studies. Alternate encodings may yield superior results. In addition, no problem-specific information was used for any of the algorithms. Problem-specific information, when available, could help all of the search algorithms examined in this study.

In the problems presented in this paper, all of the variables were encoded either with Gray-code or standard base-2 representation, as indicated with the problem. The variables were represented in non-overlapping, contiguous regions within the chromosome (solution encoding). The results reported are the best evaluations found through the search of each algorithm, averaged over at least 20 independent runs per algorithm per problem; the results for GA-SCALE and PBIL2 algorithms are the average of at least 75 runs. In the problems in which random values are assigned to problem attributes (such as the location of cities in the Traveling Salesman Problems or sizes of elements in the bin packing problems), the values are consistent across all algorithms attempted and across all 20 trials for each algorithm.

All algorithms were allowed 200,000 evaluations per run. In each run, the GA and PBIL algorithms were given 2000 generations, with 100 function evaluations per generation. In each run, the MRSH algorithms were restarted in random locations as many times as needed until 200,000 evaluations were performed. The best answer found in the 200,000 evaluations was returned as the answer found in the run.

Unfortunately, due to space limitations, the encodings for each of the problems cannot be given here; they are described in detail in [Baluja, 1995]. Brief notes about the encodings are given below, to help understand Table I, in which the *relative* results are provided. Exact results are provided in [Baluja, 1995].

To measure the significance of the difference between the results obtained by PBIL2 and GA-SCALE, the Mann-Whitney test is used. This test is a non-parametric equivalent of the standard two-sample pooled *t*-test. Results are shown in the last column of Table I.

- **TSP:** 128, 200 & 255 city problems were tried. The "sort" encoding, described in [Syswerda, 1989], was used. In this encoding, each city is assigned **$\log_2$ (#Cities)** bits. The city with the smallest value is first in the tour, the city with the second smallest is second, etc. The last problem was tried with the encoding in binary and Gray-Code.

- **Jobshop:** Two problems were tried with two encodings, the standard 10x10 and 20x5 [Muth & Thompson, 1963]. The first encoding is described in [Fang *et. al*, 1993]. The second encoding is described in [Baluja, 1995]. An additional, randomly generated, problem was also tried with the second encoding.

- **Knapsack:** In the first two problems, a unique element is represented by each bit. When a bit is set to 1, the corresponding element is included. In the third and fourth problems, there are 100 and 120 unique elements, respectively. However, there are 8 and 32 copies of each element. The number of elements of each type which are included in the solution is determined by interpreting an associated bit string, length 3 **($\log_2 8$)** bits and 5 **($\log_2 32$)** bits, into decimal, respectively.

- **Bin-Packing/Cutting Stock:** The solution is encoded in a bit string of length **M * $\log_2$N (N bins, M elements)**. Each element to be packed is assigned a sequential substring of length $\log_2$N whose value indicates the bin to place the element.

- **Neural-Network Weight Optimization:** In the first two tests, the object was to identify the parity of 7 inputs. The inputs were either 0 (represented by -0.5) or 1 (represented by 0.5). The evaluation was the sum of squares error on the 128 training examples. The network was fully connected between sequential layers. In the second two tests, eight real valued inputs were used. Two inputs represent the coordinates of a point within a square with upper left corner (ULC) of (-1.0, 1.0) and lower right corner (LRC) of (1.0, -1.0). The task was to determine whether the point fell into a square region between ULC(-0.75, 0,75), and LRC (0.75,-0.75) and outside a smaller square with ULC (-0.35, 0,35), and LRC (0.35,-0.35). 5 inputs contained random noise in the region [-1:+1]. 100 uniformly distributed examples were used. In total,

both networks had 8 inputs (including bias unit), 5 hidden units, and 1 output; this created 46 connections.

- **Numerical Function Optimization:** In the first and second problems, the variables in the first portions of the solution string have a large influence on the quality of the rest of the solution; small changes in their values can cause large changes in the evaluation of the solution. In the third problem, each variable can be set independently. The second problem has more solutions than the first. Again, the exact problems can be found in [Baluja, 1995].

Table I: Summary of Empirical Results - Ranks (1=best, 7=worst).

	Encoding Length (bits)	MRSH 1	MRSH 2	MRSH 3	PBIL 1	PBIL 2	SGA	GA Scale	MRSH BEST	PBIL BEST	GA BEST	Confidence Level (GA-Scale ≠ PBIL2)
TSP 128 city (binary)	896	6	3	4	2	1	7	5		●		> 99%
TSP 200 city (binary)	1600	5	4	3	2	1	7	6		●		> 99%
TSP 255 city (binary)	2040	5	1	2	4	3	7	6	●			> 99%
TSP 255 city (Gray-Code)	2040	5	1	2	4	3	7	6	●			> 99%
Jobshop 10x10 (Encoding 1)	500	7	5	6	2	1	4	3		●		> 99%
Jobshop 20x5 (Encoding 1)	500	7	6	5	2	1	4	3		●		> 99%
Jobshop 10x10 (Encoding 2)	700	7	5	6	3	1	4	2		●		93%
Jobshop 20x5 (Encoding 2)	700	7	5	4	2	1	6	3		●		> 99%
Jobshop 20x5 (Encoding 2)	700	7	5	4	2	1	6	3		●		> 99%
Knapsack (512 elem., 1 copy)	512	5	7	6	2	1	4	3		●		> 99%
Knapsack (2000 elem., 1 copy)	2000	4	5	6	1	3	7	2		●		> 99%
Knapsack (100 elem., 8 copies)	300	4	5	6	3	2	7	1			●	Not Available
Knapsack (120 elem., 32 copies)	600	4	5	6	2	1	7	3		●		> 99%
Bin Packing (32 bins, 128 elem.)	640	6	5	6	2	1	4	3		●		> 99%
Bin Packing (16 bins, 128 elem.)	512	6	7	5	3	1	2	4		●		Not Available
Bin Packing (4 bins, 256 elem.)	512	6	7	2	4	3	5	1			●	> 99%
Bin Packing (2 bins, 512 elem.)	512	5	7	2	3	1	4	6		●		Not Available
Neural Net PARITY 7 (binary)	368	5	5	7	2	1	4	3		●		> 99%
Neural Net PARITY 7 (gray)	368	5	6	7	2	1	3	4		●		> 99%
Neural Net SQUARE (binary)	368	5	6	7	2	1	3	4		●		> 99%
Neural Net SQUARE (gray)	368	4	2	7	1	3	5	6		●		> 99%
F1 (Encoded in Binary)	900	5	6	7	3	1	2	4		●		> 99%
F1 (Encoded in Gray Code)	900	5	6	7	2	1	3	4		●		> 99%
F2 (Encoded in Binary)	900	5	6	7	2	1	4	3		●		> 99%
F2 (Encoded in Gray Code)	900	5	4	6	2	1	7	3		●		> 99%
F3 (Encoded in Binary)	900	6	5	7	2	1	4	3		●		> 99%
F3 (Encoded in Gray Code)	900	1	1	1	5	4	7	6	●			> 99%
TOTAL (27 Problems)									3	22	2	

4. Summary

This paper has presented results on many problems. From these, it is evident that algorithms which are simpler than standard GAs can perform comparably to GAs, on both small and large problems. Other studies have also shown this for various sets of problems [Juels & Wattenberg, 1994][Forrest & Mitchell, 1993][Mitchell *et al.*, 1994], etc. In studies analyzing the performance of GAs on particular problems, these results suggest that analyses should include comparisons not only to other GAs, but also to other simpler methods of optimization before a benefit is claimed in favor of GAs. This study did not include tech-

niques such as Simulated Annealing or Tabu Search, which should be included in the future.

It is interesting to note that the PBIL algorithm, which does not use the crossover operator, and redefines the role of the population to one which is very different than that of a GA, outperforms the GAs on the majority of the problems. PBIL and GAs both generate new trials based on statistics from a population of prior trials. The PBIL algorithm explicitly maintains these statistics, while the GA implicitly maintains them in its population. The GA extracts the statistics by the selection and crossover operators. Comparisons between the two algorithms can be found in [Baluja & Caruana, 1995].

A GA with different mechanisms, such as non-stationary mutation rates, local optimization heuristics, parallel subpopulations, specialized crossover, or larger operating alphabets, may perform better than the GAs explored here. It should be noted, however, that all of these extensions, with the exception of specialized crossover operators, can be used with PBIL with few, if any, modifications.

It is incorrect to say that one procedure will always perform better than another. It has been shown that the average expected performance of all black-box optimization procedures is the same when all possible problems are considered [Wolpert & Macready, 1994]. Therefore, the results presented in this paper must be taken with caution. The results *do not* indicate that PBIL will always outperform a GA. Rather, the results indicate that algorithms, like PBIL, which are much simpler than even the simplest GAs, can outperform standard GAs on many problems of interest.

Acknowledgments

This work started from discussions with Ari Juels at UC - Berkeley. Thanks are extended to Rahul Sukthankar, Kaari Flagstad, and the Optimization Group at CMU-CS/RI for comments on drafts of this paper. This work was started while the author was supported by a National Science Foundation Graduate Fellowship. He is currently supported by a graduate student fellowship from the National Aeronautics and Space Administration, administered by the Lyndon B. Johnson Space Center. The views and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies, either expressed or implied, of NSF, NASA, or the U.S. Government.

References

Baluja, S. (1995) "An Empirical Comparison of Seven Iterative and Evolutionary Function Optimization Heuristics," CMU-CS-95-193. Available via. anonymous ftp at: reports.adm.cs.cmu.edu or http://www.cs.cmu.edu/~baluja.

Baluja, S. & Caruana, R. (1995) "Removing the Genetics from the Standard Genetic Algorithm", in A. Prieditis & S. Russel (ed.) *Machine Learning: Proceedings of the Twelfth International Conference*. Morgan Kaufmann, San Francisco, CA.

Baluja, S. (1994) "Population-Based Incremental Learning: A Method for Integrating Genetic Search Based Function Optimization and Competitive Learning". Carnegie Mellon University. Technical Report. CMU-CS-94-163.

De Jong, K. (1975) *An Analysis of the Behavior of a Class of Genetic Adaptive Systems*. Ph.D. Dissertation.

De Jong, K. (1993) "Genetic Algorithms are NOT Function Optimizers". In Whitley (ed.) *FOGA-2 Foundations of Genetic Algorithms-2*. 5-17. Morgan Kaufmann Publishers. San Mateo, CA.

Eshelman, L.J. & Schaffer, D. (1993) "Crossover's Niche". In Forrest (ed). (ICGA-5) *Proceedings of the Fifth International Conference on Genetic Algorithms*. 9-14. Morgan Kaufmann Publishers. San Mateo, CA.

Fang, H.L, Ross, P., Corne, D. (1993) "A Promising Genetic Algorithm Approach to Job-Shop Scheduling, Rescheduling, and Open- Shop Scheduling Problems". In Forrest, S. *ICGA-5.Proceedings of the Fifth International Conference on GAs.*

Forrest, S. and Mitchell, M (1993) "What Makes a Problem Hard for a Genetic Algorithm" *Machine Learning* 13. 2 & 3, 285-319.

Goldberg, D.E. (1989) *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley.

Holland, J. H. (1975) *Adaptation in Natural and Artificial Systems*. Ann Arbor: The University of Michigan Press.

Juels, A. (1996, expected) Ph.D. Thesis, University of California at Berkeley. In Progress.

Juels, A. & Wattenberg, M. (1994) "Stochastic Hillclimbing as a Baseline Method for Evaluating Genetic Algorithms," UCBerkeley. CSD-94-834.

Mitchell, M., Holland, J. & Forrest, S. (1994) "When will a Genetic Algorithm Outperform Hill Climbing" *Advances in Neural Information Processing Systems 6*, 1994. Cowan, Tesauro, Alspector (eds). Morgan Kaufmann Publishers. San Francisco, CA.

Muth, J. & Thompson, G.L., (1963) *Industrial Scheduling*, Prentice Hall International. Englewood Cliffs, NJ.

Syswerda, G. (1989) "Uniform Crossover in Genetic Algorithms". In *Proceedings of the Third International Conference on Genetic Algorithms and their Applications*, 2-9. J.D. Schaeffer, ed. Morgan Kaufmann.

Wolpert, D. & Macready, W. (1994) "No Free Lunch Theorems for Search," Santa-Fe Institute TR. SFI-TR-95-02-10.