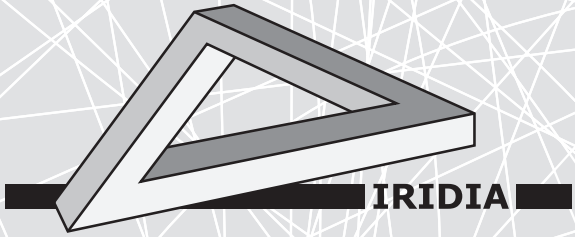

Université Libre de Bruxelles

Institut de Recherches Interdisciplinaires
et de Développements en Intelligence Artificielle

Preprocessing Fitness Functions

Yossi Borenstein and Christophe Philemotte

IRIDIA – Technical Report Series

Technical Report No.
TR/IRIDIA/2008-003

January 2008

IRIDIA – Technical Report Series
ISSN 1781-3794

Published by:
IRIDIA, *Institut de Recherches Interdisciplinaires et de Développements en Intelligence Artificielle*
Université Libre de Bruxelles
Av F. D. Roosevelt 50, CP 194/6
1050 Bruxelles, Belgium

Technical report number TR/IRIDIA/2008-003

Preprocessing Fitness Functions

Yossi Borenstein yboren@essex.ac.uk
Department of Computer Science, University of Essex, Colchester, U.K.
Christophe Philemotte cphilemo@iridia.ulb.ac.be
IRIDIA, CoDE, Université Libre de Bruxelles, Brussels, Belgium

17 January 2008

Abstract

Genetic algorithms, Evolutionary strategies and many variants of local search algorithms are being applied directly on a fitness function. Such direct search algorithms require minimal knowledge of a problem and can be applied to a wide variety of different domains. This paper suggests to apply a simple mapping on the fitness function – which we refer to as a "preprocessing" step. Any direct search algorithm is then applied on the modified function. We demonstrate empirically that our method, when applied to the Maximum Satisfiability Problem (MAXSAT) and Travelling Salesman Problem (TSP), improves the performance of a (1+1) EA.
keyword: Fitness Landscape, Preprocessing, MAXSAT, TSP, Selection

1 Introduction

Randomized search heuristics (RSHs) are a class of general purpose, stochastic algorithms which are designed to solve (optimization) problems of which only partial knowledge is known [5]. RSHs are useful mainly in the so called *black box scenario*, when there is either not enough knowledge on the problem, or alternatively, when the problem is too complicated and so, it is difficult to design a problem-specific algorithm [18].

In this scenario, rather than considering the problem directly, one tries to optimize a fitness function $f : X \to \mathbb{R}$ in which solutions $x \in X$ are mapped into corresponding fitness values [16]. RSHs sample iteratively the search space until an optimum is found (or the computational resources are exhausted). Ideally, properties of the fitness function (e.g. gradient) can be used in order to make the search faster. For example, rather than sampling solutions "blindly" one can estimate either the direction or the actual position of an optimum. This can be done in some conditions (e.g., [15]), however, it is usually computationally expensive, it cannot be precise in the general case [19] and finally, especially for discrete spaces, it is difficult to make such predictions.

RSHs use, instead, two main heuristics, a *selection scheme* which is used to select "promising" solutions and a *move operator* which, given a multiset of X, is used to generate a "similar" multiset. These two heuristics are used iteratively until an optimum is found (or resources are exhausted).

The selection scheme is usually defined directly over the fitness function. Generally, solutions are sampled proportionally to either their fitness value or to the relative rank of their fitness in relation to a sample. Solutions with high fitness are believed to represent a "good area" of the search space – i.e. it is assumed that the move operator, when applied on solutions with high fitness, will generate new solutions with relatively high fitness.

While this might be true is some cases, it is quite clear that solutions with higher fitness do not always "lead" to an optimum. Methods like simulated annealing [9] for example, explicitly select (with some probability) solutions with lower fitness as part of their strategy. This is a common strategy which account for the fact that real world problems usually contain many local optima and in many cases are highly symmetric [13].

We would like to test whether the following, simple method can increase the "reliability" of fitness as an indication for the position of an optimum. Rather than optimizing the original fitness function, we optimize a variant of the function in which the fitness of each solution is replaced with the average fitness of its k-neighborhood.

The intuition is quite straightforward, if we consider a fitness as an "evidence" for an area with potentially good solutions or a "path" which is likely to lead to an optimum, then we can assume that the more evidence there is (i.e., the higher the average of the neighborhood), the more likely our assumption to be true. In the context of *fitness landscape* [12], clearly, by replacing each solution with the average fitness of its neighborhood, the landscape becomes smoother. Measures like fitness-distance correlation [8] are likely to give higher value.

The fitness landscape is what RSHs are looking at. The fitness function only gives a very punctual information about the candidate solution. RSHs can be viewed as observers with blinkers. In *Gestalt* thinking, the perception is a combination of individual sensations, in which the environment is perceived in terms of its structure, through related macroscopic properties [1]. In [11], the developed *Gestalt* genetic algorithm detects promising area in the search space. The areas are evaluated by average over a sampling of their elements. In a same vein, the average fitness of candidate solution neighborhood respects that underlined concept of *Gestalt* thinking.

From another perspective, one can think of this method as a *preprocessing* step. If we think of the fitness function as an *image* where the topology is defined by the neighborhood structure and the "color" by the value of the fitness function, we can try to adopt tools from image analysis for the purpose of optimization. Section 2 defines formally our framework and considers some basic "filters" that one can apply as a preprocessing step on the fitness landscape. In the context of image analysis, the method which will be used in the paper can be thought of as a convolution by "moving window" in which the convolution mask assigns the same weight ($\frac{1}{|N(x)|}$) to each neighbor. In section 3 we test the effect of the preprocessing step on the performance of a (1+1) EA on the MAXSAT problem. Section 4 makes similar investigation for the TSP problem. We conclude in sections 5 (discussion) and 6 (conclusion).

2 Preliminaries

Let $f : X \to \mathbb{R}$ (where X is finite) and $d : X \times X \to \mathbb{N}$ a distance function defined over X. We are looking for a simple transformation of f which can make it "easier" for RSHs.

Naturally, such transformation does not exist in a *general* sense. The no-free-lunch theorems [19] clearly indicate that, considering a family of functions which is closed under permutation, *all* search algorithms have the same performance. This includes a search algorithm which, at the first stage, "transforms" the function and only then optimizes it. The situation might be different, though, for real world problems. As a rigorous definition of real world problems does not exist, in this paper, we will consider explicitly two NP-hard problems: MAXSAT and the TSP.

Let $g(x, m)$ be a measure of a simple statistic over the m-neighborhood of x. That is, $g(x, m) : \{y|d(y, x) = m\} \to \mathbb{R}$. For example, it can measure:

- The maximum fitness of the neighborhood:

$$g_{max}(x, m) = \max(f(y)|d(x, y) = m)$$

- The number of solutions in the neighborhood with a fitness greater than x:

$$g_{vote}(x, m) = \sum_{d(x,y)=m} \delta(f(y) > f(x))$$

- The average fitness of the neighborhood:

$$g_{avg}(x, m) = avg\left(\sum_{d(x,y)=m} f(y)\right)$$

Note that $f(x) := g(x, 0)$. In these definitions, the m-neighborhood of x is the set of y that are at an exact distance m from x: $d(y, x) = m$. If $m \neq l$, the m-neighborhood and l-neighborhood do not intersect. Let the inclusive m-neighborhood denote the union of all l-neighborhood where $l \leq m$. Over this inclusive m-neighborhood of x, we define $h(x, m)$ another measure of a simple statistic such as the average fitness : $h_{avg}(x, m) = avg\left(\sum_{d(x,y)\leq m} f(y)\right)$.

All the examples above are different variants of the same idea. We replace the assumption that: "a solution is a good indicator to the position of an optimum if it has a higher fitness" with, for example, "a solution is a good indicator to the position of an optimum if the average fitness of its neighbors is high". In this paper we consider only the average example: we test the hypothesis that g_{avg} (for MAXSAT) and h_{avg} (for TSP) is easier than f for a (1+1) EA. From this point on, g and h are assumed to be g_{avg} and h_{avg}.

Since the actual neighborhood, for some representations and operators, can be quite large, we will use a sample of the neighborhood instead. $g(x, m, n)$ denotes the average of n solutions drawn uniformly at random from the m-neighborhood of x. In the case of inclusive neighborhood, we implement $h(x, m, n)$ as n random walks of exactly m steps (this way we bias the sample towards the fitness of closer neighbors [1]).

3 MAXSAT

Let $C = \{c_1, c_2, ..., c_m\}$ be a set of m clauses that involve n Boolean variables $X = \{x_1, x_2, ..., x_n\}$. Each clause is a disjunction of k literals, that is: $c_i = \bigcup_k l_{i,k}$ where each literal $l_{i,k}$ is either a variable x_i or its negation. A clause is satisfied if at least one of its literals is satisfied. The objective of the Boolean Satisfiability problem (SAT) is to decide whether there is an assignment for X such that all the clauses are satisfied. MAXSAT is the optimization variant of SAT. The objective is to find an assignment which *maximizes* the number of satisfied clauses.

The SAT/MAXSAT problems are the subject of an extensive investigation. Many different algorithms/heuristics were and are being developed either independently or as a part of two main competitions which are dedicated solely for the SAT problem [7]. It is important to emphasize that in this paper we use the MAXSAT as an interesting problem on which we can test our hypothesis. We do not try, at this stage, to challenge the state of the art.

In the experiments reported below the size of the instances varies, however, in all the tests we focused on hard 3-SAT problems, that is the clauses to variable ratio was always 4.26 [10].

3.1 A First Order Indication

In a first experiment we wanted to test whether or not g can increase the reliability of the fitness function as an indicator to the position of optima. We generated random instances of the problem (20 variables, 84 clauses) until we obtained 100 satisfiable formulas. For each instance we run an exhaustive search in order to locate all the global optima, $\{x^*\}$. We drawn a sample of 10,000 random solutions. For each solution, x, we selected uniformly at random a neighbor, x'. We then checked whether the solution with the higher fitness is indeed closer to an optimum, i.e., whether one of the following holds:

- $f(x_1) > f(x_2)$ and $d(x^*, x_1) \leq d(x^*, x_1)$
- $f(x_2) > f(x_1)$ and $d(x^*, x_1) \geq d(x^*, x_1)$

When there were more than one optima we selected the optimum which gave the minimal distance.

We considered 5 different ways of measuring the fitness:

1. $f(x)$ (the original fitness function)

[1] Recall that the number of neighbors increases with the distance to a solution. By considering n random walks of exactly m steps, we sample n solutions from each distance. Since the number of neighbors at distance, say, 1, is much smaller than the number of neighbors at distance 6, the accuracy of of the estimation for distance 1 will be much higher.

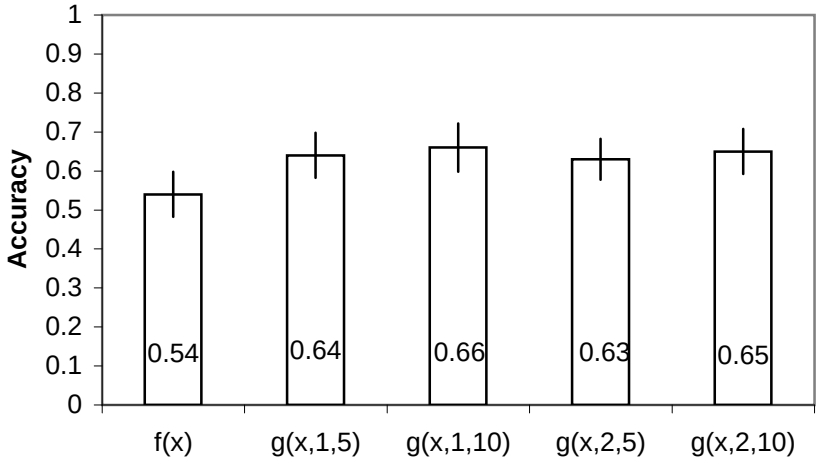

Figure 1: The fraction of trails (out of 10,000 trials) in which solutions with higher fitness are closer to an optimum.

2. $g(x, 1, 5)$ (the average of 5 solutions at distance 1)
3. $g(x, 1, 10)$ (the average of 10 solutions at distance 1)
4. $g(x, 2, 5)$ (the average of 5 solutions at distance 2)
5. $g(x, 2, 10)$ (the average of 10 solutions at distance 2)

For each case, we measured the fraction of trials (out of 10,000) in which the condition above was true.

Figure 1 shows the average and standard deviation (of the 100 instances) which were obtained for the five cases. All the variants of g give better first-order indication to the position of an optimum. In particular, while the original function was accurate only in 0.54 of the trials, $g(x, 1, 10)$ was accurate in 0.66 (we run a t-test with $p \ll 0.01$).

It was of a particular interest to check whether these results change as a function of the fitness value. Presumably, the distribution of *good* solutions in the neighborhood of *bad* solutions (e.g., solutions with fitness below the median) is different than the distribution of solutions in the neighborhood of good solutions. In order to check this we grouped the 100*10,000 trials (obtained before) according to the fitness of the first solution. Doing so, we received about 20 samples, one for each fitness value from 63 to 81. The number of solutions in each sample differs [2], however, we considered samples with at least 100 pairs of solutions.

Figure 2 shows that $g(x, 1, 10)$ is consistently better than $f(x)$. Interestingly, the accuracy for both $f(x)$ and $g(x, 1, 10)$ seems to decrease as the fitness get closer to the median. We do not know at this stage whether it is an averaging effect (as the sample in these solutions was 100 times larger than the samples of solutions in the "edge") or some trend which is related to the MAXSAT problem.

3.2 Results For the (1+1) EA

The results above support the hypothesis that the transformation of the MAXSAT fitness function can make it easier. However, they were obtained only as a first-order approximation. In practice, it might be the case that a solution which increases the distance to an optimum (which was considered negative) is a part of a "path" which eventually leads to an optimum.

[2] As the fitness distribution of MAXSAT is normal, sampling uniformly the space, the number of solutions with a median fitness is much larger than the number of higher or lower fitness values.

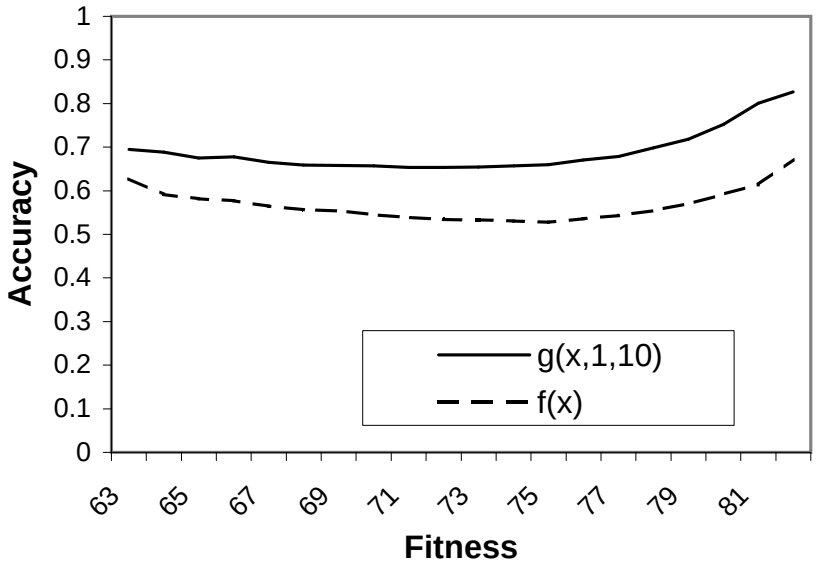

Figure 2: The fraction of trails in which solutions with higher fitness are closer to an optimum, sorted by fitness value of the first solution.

In order to test whether this is the case we used the following (1+1) EA to optimize instances with 50 variables and 210 clauses.

Select $x \in X$ uniformly at random
while notImprovedCount ¡ maxNotImprovedSteps **do**
  Select $x' \in N(x)$ uniformly at random
  if $f(x') > f(x)$ **then**
    $x := x'$
    notImprovedCount := 0
  else
    notImprovedCount := notImprovedCount +1
  end if
end while

Whenever a local improvement was not obtained within 50 time steps (i.e., maxNotImproved-Steps=50), the algorithm was restarted. This was repeated until 10,000 fitness evaluations occurred, in which case the *trial* (or, run) was terminated. We say that the trial was successful if a solution which satisfies the formula was found.

At each test, we sampled randomly 200 instances and measured the number of successful trials for $f(x)$ (the actual instance) and a $g(x, 1, 50)$ variant. Figure 3 represents the result of 10 such tests. In 8/10 tests the $g(x, 1, 50)$ was strictly easier than $f(x)$. In the other 2 tests the performance of the (1+1) EAs over the two functions was the same.

So far we did not consider the computational cost of measuring the average. Clearly, in practice, we sampled up to 50 times mores solutions in the case of $g(x, 1, 50)$ as opposed to $f(x)$. In order to make the comparison fair, we did not consider solutions which were not chosen by the (1+1) EA. For example, if a neighbor of an optimum was chosen, and as a part of measuring its fitness, the actual optimum was sampled, we did not count it as a success. Only if the actual optimum was chosen (i.e., x' is an optimum) then we marked the run as successful.

The objective so far was simply to test the hypothesis that $g(x, 1, 50)$ is easier than $f(x)$. While in some tests the impact of calculating the average was not big (i.e., only 2 or 3 more trials were

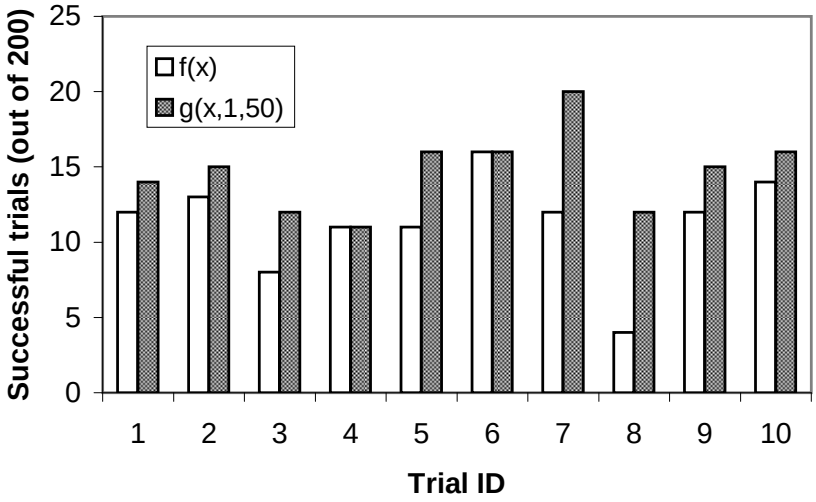

Figure 3: 10 tests of a (1+1) EA on original and modified MAXSAT instances.

successful), the (1+1) EA was consistently better on $g(x, 1, 50)$ in comparison to $f(x)$.

In order to use this method in practice one has to consider the trade-off between the size of the sample (i.e., the accuracy of estimating g_{avg}) and the computational cost. We are working at the present on efficient methods to implement the sampling (for example, by excluding all the common neighbors of different solutions). Once this is done we can test whether or not it is worthwhile to use this method for the MAXSAT problem in practice. In the next section we show that this is the case for the TSP problem.

4 TSP

In combinatorial optimization, the TSP is certainly one of the most studied, popular and representative problems. The objective of the TSP problem is to find the shortest tour through a set of towns which have to be visited exactly once. It is a minimization problem. Since it is a NP-hard problem [6], most of the efforts have been focused on the design of algorithms that can find satisfactory solutions in a decent time. The current best performing algorithms are, to some extent, a combination of global and local search methods [14, 17]. As in the MAXSAT case, we use the TSP as an interesting problem on which we can test whether the preprocessing step is worthwhile. This time we use a different variant of the (1+1) EA. Moreover, in order to account for the extra computational cost of the preprocessing step, we measure performance as the best objective value obtained in a given, fixed, *time* (irrespectively of the number of fitness evaluations).

In the experiments reported below, we consider TSP instances with 100 towns. We distinguish between two classes: uniform points in the plane and clustered points. Both classes are generated by the Benchmark TSPLIB Instance Generation Codes [3] from the 8th DIMACS Implementation Challenge. For a given class of instances, the optimal tours length are of the same order of magnitude.

4.1 Algorithmic considerations

The representation of TSP is based on permutations. For permutations, there are various mutation operators [2]. We will use one of the most common and efficient mutation operators: the Exchange Mutation (EM) [3]. As mentioned in Section 2, the evaluation, $h(x, m, n)$, of a candidate solution

[3]The source can be obtained on
`http://www.research.att.com/~dsj/chtsp/download.html`

x is the average of n random walks of m steps:

$$h(x, m, n) := \frac{f(x) + \sum_{k=1}^{n} \sum_{l=1}^{m} f(x_l^k)}{1 + nm}$$

where x_l^k is the candidate solution obtained after l different mutations in the walk k. Let σ denote the EM operator. We used the following (1+1) EA:

Select $x \in X$ uniformly at random
while spent time ¡ max time **do**
  Mutate $x' := \sigma(x)$
  if $h(x', m, n) < h(x, m, n)$ **then**
    $x := x'$
  end if
end while

In order to measure the average fitness (h) in an efficient way, we used the following procedure. Consider the candidate solutions x and $x' = \sigma(x)$. Since they differ in the place of 2 towns, the fitness value $f(x')$ can be computed from $f(x)$ by subtracting the length of the 4 old edges that connected the 2 towns and adding the length of the 4 new edges that connect the 2 towns in their new places. This means that we do not need to compute, for each mutation, the fitness value from the beginning.

4.2 Methodology and Results

The experimental methodology consists of two phases. First, we use a racing method to find, in a given computation time, the best number of walks (neighborhood size) and the corresponding sample size (n, m). The best tuple [4] (n, m) is selected according to corresponding performance of the (1+1) EA. In the second phase, we run several experiments using the (1+1) EA with the best parameters.

In the first phase, we used the F-Race racing algorithm [5] [4]. This racing algorithm tests different configurations (n, m) over different instances. For each instance, the (1+1) EA is run for each configuration. From the obtained results, the configurations are ranked with a Friedman test. Considering a given confidence level, the Friedman test gives the significantly bad configurations (i.e. with a corresponding p-value). These significantly bad configurations are deleted from the configurations pool and not considered in the next steps of the racing algorithm.

Table 4.2 summarizes the different experiments of the first phase. Note the relative score $\Delta s := \frac{s - s_{opt}}{s_{opt}}$ where s is the obtained tour length and s_{opt} the optimal tour length. For each experiment, the relative scores obtained over all instances are shown as boxplots (Fig. 4, 5, 6, 7, 8 and 9).

The best parameters, following the results of the first phase, are a walk of length 1, and a sample of size 1 (i.e., $h(x, 1, 1)$). This means that fitness is evaluated as the average of the original fitness value and the fitness of one, randomly chosen neighbor. But some interesting facts have been observed. In general, it seems that the (1+1) EA (with $f(x)$) is trapped relatively quickly in local optimum. For clustered instances, the tuples around $(100, 10)$ give better results in the beginning of the search than the $(1, 1)$.

In the second phase we generated one clustered instance and one uniform instance. We tested the performance of the (1+1) EA using the best tuples obtained in the first phase and compared the results with the performance obtained over the default fitness function, f. Each run lasted 600s. The relative scores of 50 different runs are plotted in each case (see Fig. 10 and 11).

[4] Note that $f(x) := g(x, 0) = h(x, 0)$.
[5] A R package of F-Race is available on
`http://cran.r-project.org/src/contrib/Descriptions/race.html`

Experiment	Parameters		Instances		Winner
(Nbr of (n, m))	conf. lvl (%)	time (s)	nbr	class	(n, m)
1 Fig. 4 (40)	95	60	20	uni.	$(1, 1)$
2 Fig. 5 (55)	95	60	20	uni.	$(1, 1)$
3 Fig. 6 (2)	99	600	30	uni.	$(1, 1)$
4 Fig. 7 (40)	95	60	20	clu.	$(100, 10)$
5 Fig. 8 (55)	95	60	20	clu.	$(90, 12)$
6 Fig. 9 (8)	99	600	30	clu.	$(1, 1)$

Table 1: The tuning experiments with Birattari's race algorithm based on a Friedman test [4]. Two instances classes are considered: uniform points in the plan (uni.) and clustered points (clu.).

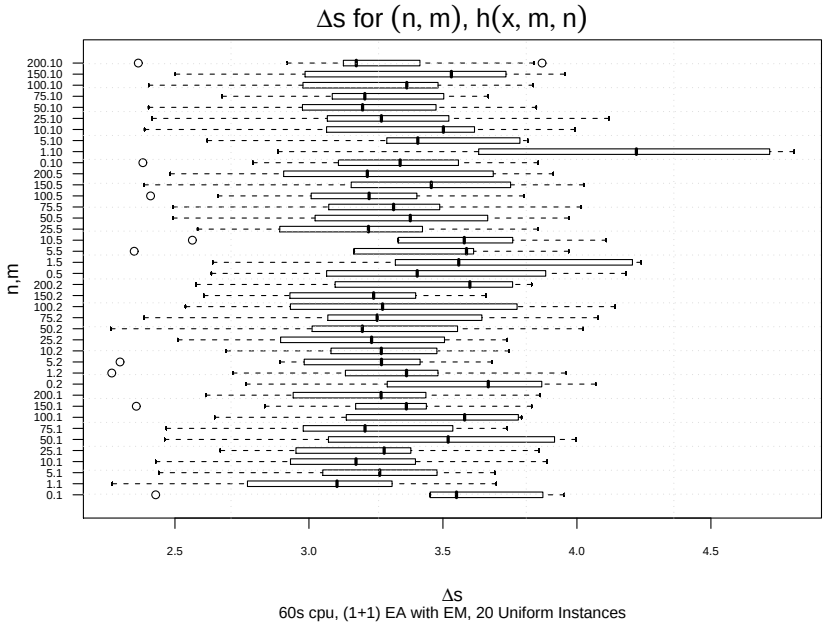

Figure 4: Tuning experiment 1: 40 (n, m), confidence level of 95%, allocated computation time of 60s, 20 uniform instances. The winner is $(1, 1)$. The relative scores Δs for each configuration (n, m) are plotted as boxplot.

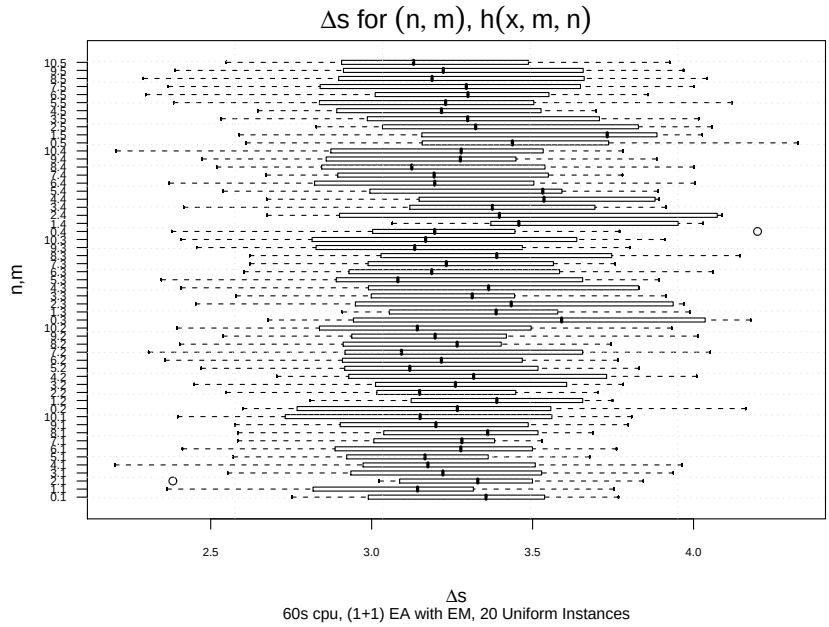

Figure 5: Tuning experiment 2: 55 (n, m), confidence level of 95%, allocated computation time of 60s, 20 uniform instances. The winner is $(1, 1)$. The relative scores Δs for each configuration (n, m) are plotted as boxplot.

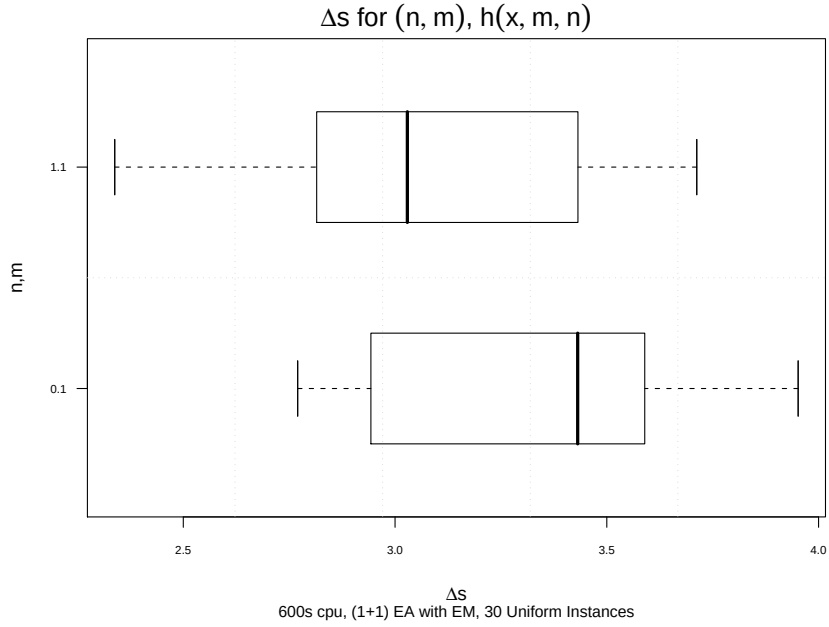

Figure 6: Tuning experiment 3: 2 (n, m), confidence level of 99%, allocated computation time of 600s, 30 uniform instances. The winner is $(1, 1)$. The relative scores Δs for each configuration (n, m) are plotted as boxplot.

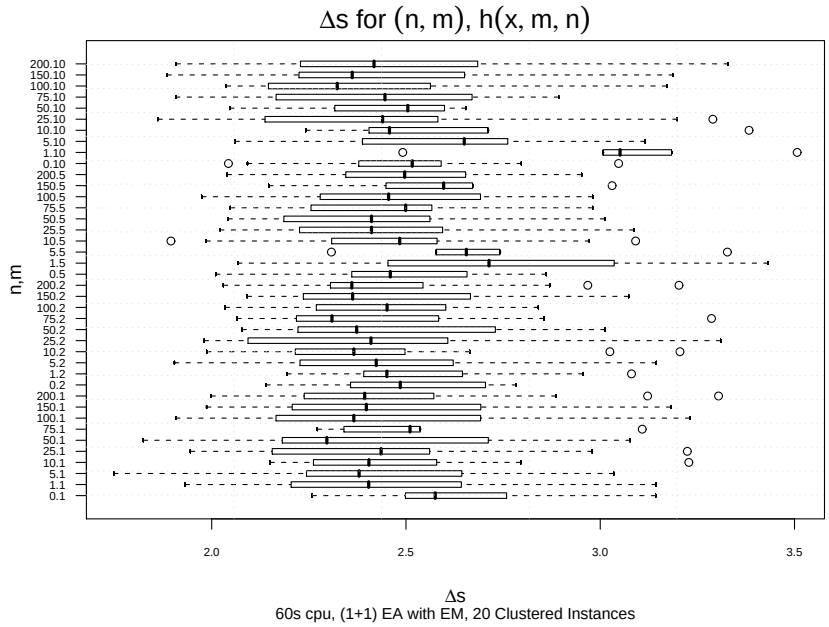

Figure 7: Tuning experiment 4: 40 (n, m), confidence level of 95%, allocated computation time of 60s, 20 clustered instances. The winner is $(100, 10)$. The relative scores Δs for each configuration (n, m) are plotted as boxplot.

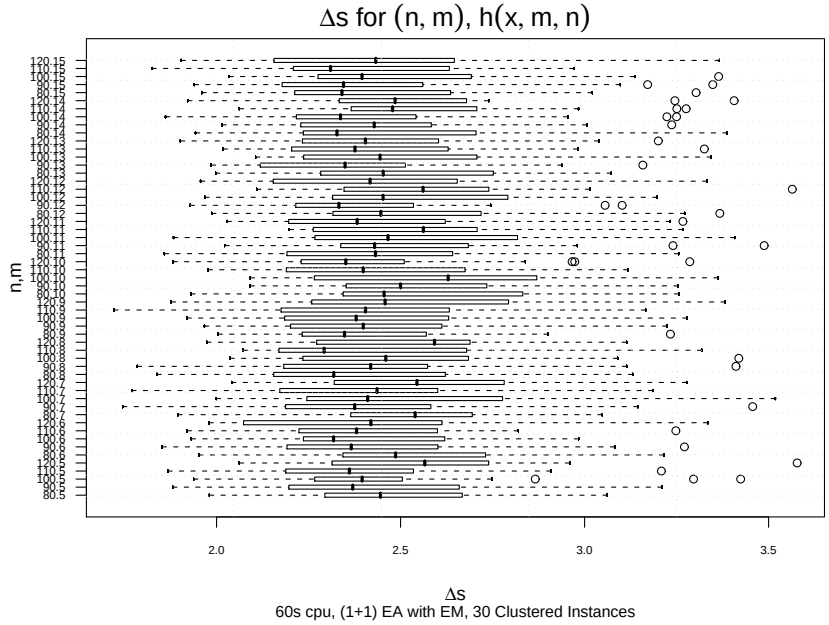

Figure 8: Tuning experiment 5: 55 (n, m), confidence level of 95%, allocated computation time of 60s, 20 clustered instances. The winner is $(90, 12)$. The relative scores Δs for each configuration (n, m) are plotted as boxplot.

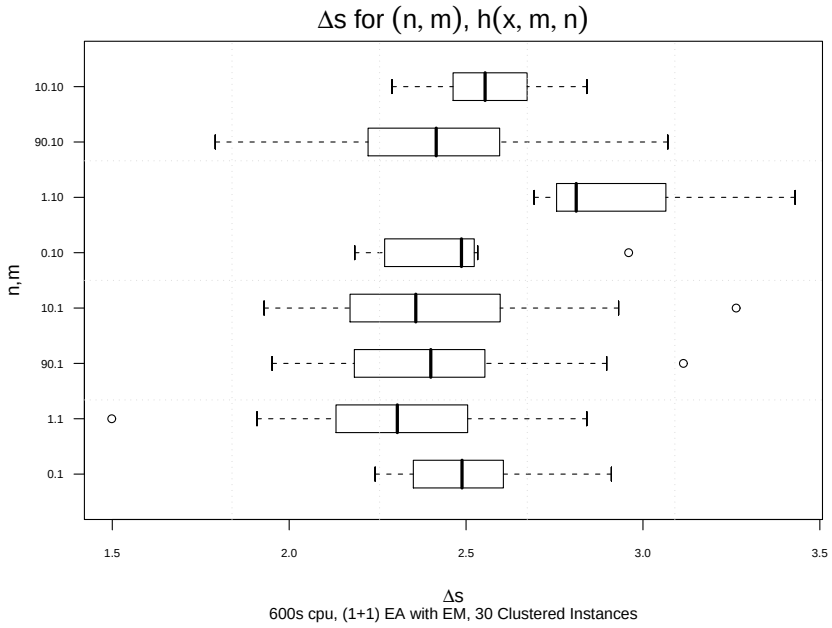

Figure 9: Tuning experiment: 8 (n, m), confidence level of 99%, allocated computation time of 600s, 30 uniform instances. The winner is $(1, 1)$. The relative scores Δs for each configuration (n, m) are plotted as boxplot.

For the uniform instance, a Mann-Whitney U test with the alternative hypothesis $H_1 := \Delta s_{f(x)} > \Delta s_{h(x,1,1)}$ gives a p-value $< 2.2\,10^{-16}$. $h(x, 1, 1)$ is significantly easier than $f(x)$. For clustered instance, a Kruskal-Wallis test gives a p-value $= 3.559\,10^{-6}$. Once again, $h(x, 1, 1)$ is significantly easier. We observe that $f(x)$ gives the worse result.

5 Discussion

RSHs are usually applied directly on the fitness function. This paper suggests to use a simple preprocessing step, in which the fitness of each solution is replaced by the average fitness of its neighborhood (where the size of the neighborhood varies). In practice, due to the computational cost of measuring the average, we used a sample of the neighborhood instead. This step (sampling the neighborhood and measuring the average) was applied dynamically as the search progresses.

Section 3 investigated the effect of this method on the MAXSAT problem. We sampled pairs of solutions uniformly at random and measured the frequency of pairs in which the solution with the higher fitness value was closer to an optimum. We showed that the frequency was higher when applying the preprocessing step. This suggests that the correlation between fitness values and the distances from an optimum is higher when applying the preprocessing step.

We then measured the number of runs in which an optimum was found for a hard MAXSAT problem with 50 variables. The (1+1) EA had a better performance on 8/10 trials and the same performance in 2 trials. This supported our claim – the modified objective functions were easier than the original ones. However, we did not consider, in this experiment the computational cost of measuring the average.

Section 4 studied the potential advantage of our method on the TSP problem. This time, performance was measured as a function of time, rather than fitness evaluations. At a first stage, we run a racing algorithm to find the best parameters (size of neighborhood and sample) for the preprocessing step. Then the performance of a (1+1) EA was measured both on the original and the modified fitness function. In all the experiments, we run the algorithm for 600s. The (1+1) EA was shown to significantly perform better on the preprocessed fitness function.

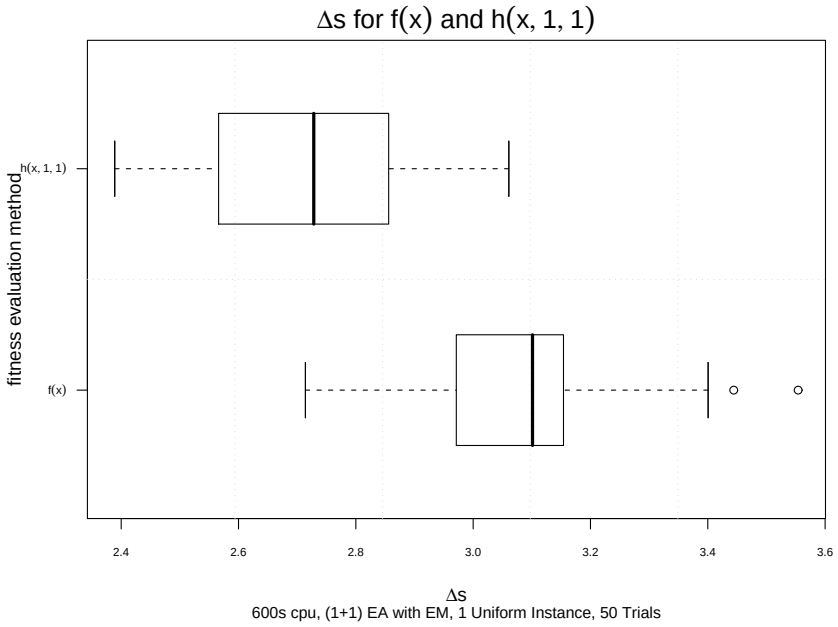

Figure 10: For one uniform instance, 50 trials of (1+1) EA using $f(x)$ or $h(x, 1, 1)$ are run. The relative scores Δs are plotted as boxplot. A Mann-Whitney U test with alternative hypothesis $H_1 := \Delta s_{f(x)} > \Delta s_{h(x,1,1)}$ gives a p-value $< 2.2\,10^{-16}$. The $h(x, 1, 1)$ significantly wins.

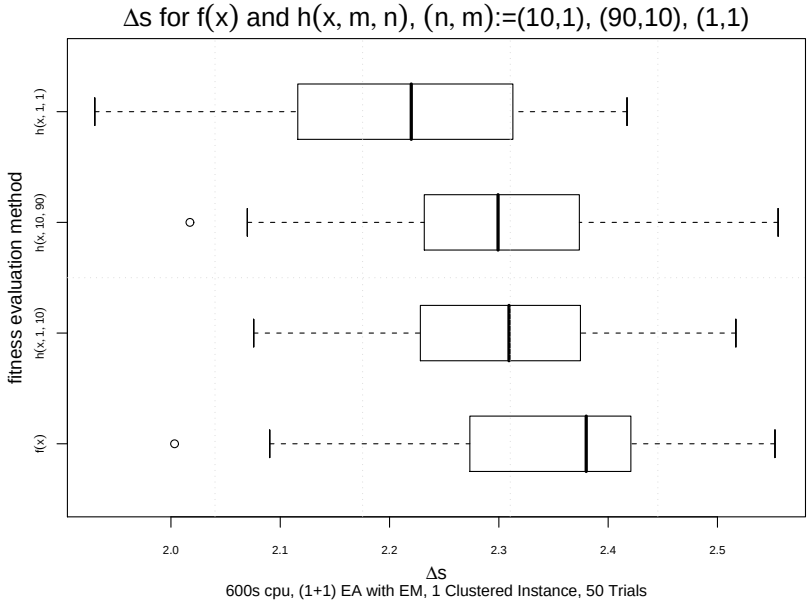

Figure 11: For one clustered instance, 50 trials of (1+1) EA using $f(x)$ or h with $(n, m) \in \{(1, 1), (10, 1), (90, 10)\}$. A Kruskal-Wallis test gives a p-value $= 3.559\,10^{-6}$. The $h(x, 1, 1)$ significantly wins.

6 Conclusion

We argued that a simple, generic, modification of a fitness function which do not require any a priori knowledge of the problem can increase the performance of RSHs. This was demonstrated for a (1+1) EA running on two NP-hard problems which use different representations (binary and permutation). In future research we plan to investigate alterative ways of modifying the fitness function (see section 2) and test their effect on other RSHs.

References

[1] M. Ash. *Gestalt Psychology In German Culture 1890 - 1967*. Cambridge University Press, 1995.

[2] T. Back, D. B. Fogel, and Z. Michalewicz, editors. *Basic Algorithms and Operators*. IOP Publishing Ltd., Bristol, UK, UK, 1999.

[3] W. Banzhaf. The molecular traveling salesman. *Biological Cybernetics*, 64(1):7–14, 1990.

[4] M. Birattari, T. Stützle, L. Paquete, and K. Varrentrapp. A racing algorithm for configuring metaheuristics. In *GECCO '02: Proceedings of the Genetic and Evolutionary Computation Conference*, pages 11–18, San Francisco, CA, USA, 2002. Morgan Kaufmann Publishers Inc.

[5] C. Blum and A. Roli. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Comput. Surv.*, 35(3):268–308, 2003.

[6] M. R. Garey, R. L. Graham, and D. S. Johnson. Some np-complete geometric problems. In *STOC '76: Proceedings of the eighth annual ACM symposium on Theory of computing*, pages 10–22, New York, NY, USA, 1976. ACM Press.

[7] H. H. Hoos and T. Sttzle. Satlib: An online resource for research on sat. In *SAT 2000*, pages 283–292. IOS, 2000.

[8] T. Jones and S. Forrest. Fitness distance correlation as a measure of problem difficulty for genetic algorithms. In *Proceedings of the 6th International Conference on Genetic Algorithms*, pages 184–192, San Francisco, CA, USA, 1995. Morgan Kaufmann Publishers Inc.

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

[10] R. Monasson, R. Zecchina, S. Kirkpatrick, B. Selman, and L. Troyansky. Determining computational complexity from characteristic /'phase transitions/'. *Nature*, 400(6740):133–137, July 1999.

[11] C. Philemotte and H. Bersini. A gestalt genetic algorithm: less details for better search. In *GECCO '07: Proceedings of the 9th annual conference on Genetic and evolutionary computation*, pages 1328–1334, New York, NY, USA, 2007. ACM.

[12] C. M. Reidys and P. F. Stadler. Combinatorial landscapes. *SIAM Rev.*, 44(1):3–54, 2002.

[13] A. Rogers, A. Prügel-Bennett, and N. R. Jennings. Phase transitions and symmetry breaking in genetic algorithms with crossover. *Theor. Comput. Sci.*, 358(1):121–141, 2006.

[14] T. Stützle. *Local Search Algorithms for Combinatorial Problems - Analysis, Algorithms, and New Applications*. PhD thesis, Department of Computer Science, Darmstadt University of Technology, 1998.

[15] S. R. Sylvain Gelly and O. Teytaud. Comparison-based algorithms are robust and randomized algorithms are anytime. *Evolutionary Computation*, page to appear, 2007.

[16] M. D. Vose. *The Simple Genetic Algorithm: Foundations and Theory*. MIT Press, Cambridge, MA, USA, 1998.

[17] T. Walters. Repair and brood selection in the traveling salesman problem. In A. Eiben, T. Back, M. Schoenauer, and H.-P. Schwefel, editors, *Proceedings of the Fifth International Conference on Parallel Problem Solving from Nature, Amsterdam, The Netherlands*, volume 1498 of *Lecture Notes in Computer Science*, pages 813–822. Springer-Verlag, 1998.

[18] I. Wegener. Randomized search heuristics as an alternative to exact optimization. In W. Lenski, editor, *Logic versus Approximation, Essays Dedicated to Michael M. Richter on the Occasion of his 65th Birthday*, volume 3075 of *Lecture Notes in Computer Science*, pages 138–149. Springer, 2004.

[19] D. Wolpert and W. G. Macready. No free lunch theorems for optimization. *IEEE Trans. Evolutionary Computation*, 1(1):67–82, 1997.