Model-Based Search for Combinatorial Optimization: A Comparative Study

Mark Zlochin[1,⋆] and Marco Dorigo[2]

[1] Dept. of Computer Science, Technion – Israel Institute of Technology, Haifa, Israel
`zmark@cs.technion.ac.il`
[2] IRIDIA, Université Libre de Bruxelles, Brussels, Belgium
`mdorigo@ulb.ac.be`

Abstract In this paper we introduce *model-based search* as a unifying framework accommodating some recently proposed heuristics for combinatorial optimization such as ant colony optimization, stochastic gradient ascent, cross-entropy and estimation of distribution methods. We discuss similarities as well as distinctive features of each method, propose some extensions and present a comparative experimental study of these algorithms.

1 Introduction

The necessity to solve $\mathcal{NP}$-hard problems, for which the existence of efficient exact algorithms is highly unlikely, has led to a wide range of heuristic algorithms that implement some sort of search in the solution space. These heuristic algorithms can be classified, similarly to what is done in the machine learning field [15] , as being either *instance-based* or *model-based*. Most of the classical search methods may be considered instance-based, since they generate new candidate solutions using solely the current solution or the current "population" of solutions. Typical representatives of this class are genetic algorithms or local search and its variants, such as, for example, simulated annealing and iterated local search.On the other hand, in the last decade several new methods, which may be classified as *model-based search* (MBS) algorithms, have been proposed. In MBS algorithms, candidate solutions are generated using a parameterized probabilistic model that is updated using the previously seen solutions in such a way that the search will concentrate in the regions containing high quality solutions.

In [20], several MBS approaches, such as *ant colony optimization* (ACO) metaheuristic [6], *stochastic gradient ascent* (SGA) [16, 12], *cross-entropy* (CE) method [18] and *estimation of distribution algorithms* (EDAs) [11], were considered within a common framework, and analysis of their similarities as well as their distinctive features was provided.

⋆ This work was carried out while the author was at IRIDIA, Université Libre de Bruxelles, Belgium.

In this paper we provide a detailed comparative analysis of these algorithms, in the context of unconstrained binary coded problems. In addition to the analytical comparison, we also present an experimental comparison of the MBS methods discussed in this paper using the MAXSAT problem as a test bed.

2 Model-Based Search

Let us consider a minimization problem$(\mathcal{S}, f)$, where $\mathcal{S}$ is the *set of feasible solutions*, f is the *objective function*, which assigns to each solution $s \in \mathcal{S}$ a cost value $f(s)$. The goal of the minimization problem is to find a feasible solution of minimal cost.

At a very general level, the model-based search approach attempts to solve this minimization problem by repeating the following two steps:

– Candidate solutions are constructed using some parameterized probabilistic model, that is, a parameterized probability distribution over the solution space.
– The candidate solutions are used to modify the model in a way that is deemed to bias future sampling toward high quality solutions.

For any algorithm belonging to this general scheme, two components, corresponding to the two steps above, need to be instantiated:

– A probabilistic model that allows an efficient generation of the candidate solutions.
– A rule for updating the model's parameters.

In this paper we restrict our attention to those problems whose solutions can be coded as unconstrained binary strings, although ACO, SGA and CE can be applied in a more general context as well. A more general discussion about these methods and the relations between them is given in [20]. All of the methods described in this paper employ the following probabilistic model for generating candidate solutions:

– Every bit position, $1 \leq i \leq n$, has an associated parameter τ_i.
– The bits, s_i, are generated independently, with $P(s_i = 1) = F(\tau_i)$[1].

The parameter vector and the resulting probability distribution over the solution space are denoted by $\mathcal{T}$ and $P_{\mathcal{T}}(\cdot)$ correspondingly. The parameters are typically initialized in such a way that the initial distribution is a uniform one.

It should be noted that several algorithms that fall within the MBS framework and that allow for dependencies between the bit positions have been described in the literature (e.g., MIMIC [3], BOA [14] or variants of ACO [17]). Some of these algorithms were reported to yield certain improvement over the simpler algorithms, which generate bits for different positions independently.

[1] In fact, all the algorithms considered in this paper, except for SGA, use simply $F(\tau_i) = \tau_i$.

However, these comparisons were performed on a basis of equal number of iterations, rather than equal computational time. On the other hand, our personal experience, as well as some results in the literature (e.g., [17]), suggest that the considerable computational overhead imposed by using a more complex model renders the "dependencies learning" algorithms uncompetitive. Consequently, these algorithms are not considered in this paper.

3 The Algorithms

In this section we give a brief description of several existing MBS algorithms and discuss the relationships among them. The reader is referred to [20] and references therein for a more detailed discussion. It should be emphasized that the probabilistic model employed by all these algorithms is the same, hence the only difference among them is in the way the parameters are interpreted and modified.

3.1 Ant Colony Optimization

One well-established approach that belongs to the MBS framework is the *ant colony optimization* (ACO) metaheuristic. In ACO algorithms, solutions are generated using stochastic procedures, called *artificial ants*, which construct them by iteratively adding solution components. The components are chosen with a probability which is a function of so called *pheromone* values associated to components. After constructing the solutions, the pheromone values associated with the components belonging to good solutions are increased. This metaheuristic has been successfully applied to the solution of numerous NP-hard problems [6] as well as to time-varying stochastic optimization problems [4]. A particular variant of this metaheuristic, called Hyper-Cube (HC) ACO [2], has been recently proposed in the context of combinatorial problems with binary coded solutions. In HC-ACO the pheromones are bounded between zero and one and the pheromone update rule can be described by the following general scheme:

HC_ACO_UPDATE

$$\tau_i \leftarrow (1-\rho)\tau_i + \rho \, \frac{\sum_{s \in S_t} Q_f(s) s_i}{\sum_{s \in S_t} Q_f(s)}. \tag{1}$$

where S_t is the sample in the t-th iteration, ρ, $0 \leq \rho < 1$, is the learning rate and $Q_f(s|S_1, \ldots, S_t)$ is some "quality function", which is typically required to be non-increasing with respect to f and is defined over the "reference set" $\hat{S}_t$.

In the considered case of unconstrained problems, the pheromones are equal to the marginal probabilities of the corresponding positions, with the bits at different positions being assigned independently[2].

[2] In a preliminary study, we have also considered using heuristic information, similarly to [17]. However, the improvement in performance was negligible, when compared to the improvement obtained with local search. Therefore, it was decided to limit our discussion in this paper to the simpler version of HC-ACO.

Different ACO algorithms may use different quality functions and reference sets. For example, in the very first ACO algorithm — Ant System [5]— the quality function was $1/f(s)$ and the reference set $\hat{S}_t = S_t$. In a more recently proposed scheme, called *iteration best update* [7], the reference set was a singleton containing the best solution within S_t (if there were several iteration-best solutions, one of them was chosen randomly). In the *global-best update* [7, 19], the reference set contained the best among all the iteration-best solutions (and if there was more than one global-best solution, the earliest one was chosen).

In $\mathcal{MAX}$–$\mathcal{MIN}$ Ant System [19], maximum and minimum pheromone trail limits were introduced. With this modification the probability to generate any particular solution is kept above some positive threshold, which helps preventing search stagnation and premature convergence to suboptimal solutions. For HC-ACO, this approach translates into the requirement that the marginal probabilities are kept within the range $[\epsilon, 1-\epsilon]$, where $\epsilon \geq 0$ is the parameter that controls the amount of exploration.

It is worth noting that, as shown in [8], for learning rate $\rho = 1$ and for a particular choice of the quality function, the HC-ACO is equivalent to the *cross-entropy method* [18].

3.2 The Stochastic Gradient Ascent Method

While all the updates described above are of a somewhat heuristic nature, the SGA method allows to derive the parameters update rule in a more principled manner [12].

The SGA method replaces the original optimization problem with the following equivalent continuous *maximization problem*:

$$\mathcal{T}^* = \underset{\mathcal{T}}{\operatorname{argmax}}\, \mathcal{E}(\mathcal{T}), \tag{2}$$

where $\mathcal{E}(\mathcal{T}) = E_{\mathcal{T}} Q_f(s)$ and $E_{\mathcal{T}}$ denotes expectation with respect to $P_{\mathcal{T}}$. This maximization problem is, in turn, tackled using *stochastic gradient ascent* [16]:

$$\mathcal{T}^{t+1} = \mathcal{T}^t + \alpha_t \sum_{s \in S_t} Q_f(s) \nabla \ln P_{\mathcal{T}^t}(s), \tag{3}$$

where S_t is the sample at iteration t.

In [8] it was demonstrated how the required gradient $\nabla \ln P_{\mathcal{T}^t}(s)$ can be calculated for a general class of probabilistic models. For the model we consider in this paper, namely binary variables without dependencies between different positions, it can be verified that the resulting parameter update rule is:

SGA_Update

$$\tau_i \leftarrow \tau_i + \alpha_t \sum_{s \in S_t} Q_f(s) \frac{F'(\tau_i)}{s_i \cdot F(\tau_i) - (1-s_i) \cdot (1 - F(\tau_i))} \tag{4}$$

In order to guarantee the stability of the resulting algorithm, it is desirable to have a bounded gradient $\nabla \ln P_{\mathcal{T}}(s)$. For this reason, the use of the "natural" representation $F(\tau_i) = \tau_i$ is inappropriate. Instead, we suggest using the logistic function $F(\tau_i) = \dfrac{1}{1+\exp(-\tau_i)}$. It can be shown that in this case the update rule becomes:

$$\tau_i \leftarrow \tau_i - \alpha_t \sum_{s \in S_t} Q_f(s) F(\tau_i) + \alpha_t \sum_{s \in S_t} Q_f(s) s_i, \tag{5}$$

hence the gradient is indeed bounded.

While the SGA method was originally formulated for iteration-independent quality functions, in [8] it was demonstrated that an alternative derivation of the SGA update through the CE method justifies the use of iteration-dependent quality functions as well. For example, one may use the indicator function $Q_f(s) = I(f(s) < \theta_t)$, where θ_t is a threshold value set to some percentile (say, lower 10%, for minimization problems) of the cost distribution at the last iteration. For this quality function, the expectation $E_{\mathcal{T}} Q_f(s)$ equals the probability of generating solutions, whose cost is below the threshold θ_t, and the threshold is modified adaptively. The update based on this function is henceforth referred to as "top-quality" update.

Similarly to HC-ACO, we may choose to bound the marginal probabilities in order to increase the amount of exploration. In order to keep the marginal probabilities between ϵ and $1-\epsilon$, with the logistic function representation described above, the pheromones should be kept in the range $[\ln(\frac{\epsilon}{1-\epsilon}), \ln(\frac{1-\epsilon}{\epsilon})]$.

3.3 Estimation of Distribution Algorithms

As already mentioned in Section 1, the classical genetic algorithms can be considered an example of the instance-based approach, in which the search is carried out by evolving the population of candidate solutions using selection, crossover and mutation operators. Recently, several new algorithms, which generate new solutions using probabilistic models instead of crossover and mutation, have been proposed within the evolutionary computation community.

In the population-based incremental learning (PBIL) algorithm [1], the population is replaced by a probability vector $\bar{p}$, with all p_i's initially set to 0.5. At every iteration a sample S is generated using the probability vector and then the probability vector is updated as follows:

```
PBIL_Update
  – $S_{top} \leftarrow$ a fixed number of lowest cost solutions from $S$,
  – for every $s \in S_{top}$
      – $p_i \leftarrow (1-\rho)p_i + \rho s_i$,
where $\rho$ is the learning rate.
```

As it can be easily seen, this update is virtually identical (up to rescaling of the learning rate) to the HC-ACO with top-quality update. In particular, in case only the best solution is used for the update, HC-ACO with iteration-best update is obtained. In [1] two additional updates were suggested. The first was intended to make use of "negative" examples, shifting the probability vector towards the best solution in the positions where it differs from the worst solution:

```
– if $s_i^{best} \neq s_i^{worst}$ then
      $p_i \leftarrow (1-\rho_{nl})p_i + \rho_{nl} s_i^{best}$,
```

where s^{best}, s^{worst} are respectively the best and the worst solutions in S and ρ_{nl} is the so-called "negative learning rate". In the second update, the probability vector was randomly perturbed, with an effect similar to that of mutation in standard GA:

```
– For every $i$, modify $p_i \leftarrow (1-\rho_{mut})p_i + \rho_{mut} d_i$, with probability $p_{mut}$.
```

where ρ_{mut} is the "mutation shift", and, for every i, the mutation direction d_i is 0 or 1, with probability 1/2 each. Both updates were performed in addition to the basic PBIL update described above.

The compact genetic algorithm (cGA) [9] was proposed as a modification of PBIL, intended to represent more faithfully the dynamics of the real GA algorithm. At every iteration, two solutions, a and b, are generated using a probability vector, and then the probability vector is updated as follows (assuming, without loss of generality, that a has lower cost):

```
cGA_Update
  – if $a_i \neq b_i$ then
      if $a_i = 1$ then $p_i \leftarrow p_i + 1/n$,
               else $p_i \leftarrow p_i - 1/n$,
    where $n$ is a parameter, equivalent to the population size in the classical GA.
```

This basic scheme can be extended to larger samples. Two variants were proposed in [9]. In the first variant, intended to simulate a tournament of size m, a sample S of size m is generated and the basic update above is used for every pair in the set $\{(s^{best}, b) | b \in S, b \neq s^{best}\}$. In the second variant, a "round-robin tournament" is simulated, that is, the basic update is used for every pair of solutions from the sample.

It can be shown that the update for "tournament of size m" cGA can be written as:

$$p_i \leftarrow p_i + \rho \sum_{s\in S} Q(s) s_i - \frac{\rho}{m} \sum_{s \in S} s_i, \tag{6}$$

where $\rho = \frac{m}{n}$ and

$$Q(s) = \begin{cases} 1\ , & s = s^{best} \\ 0\ , & \text{otherwise.} \end{cases} \tag{7}$$

For the "round-robin tournament" cGA, it can be shown that the update can also be described by (6), with $\rho = \frac{m(m+1)}{n}$ and $Q(s) = \frac{2 \cdot \mathrm{rank}(s)}{m(m+1)}$ (the highest rank, m, is assigned to s^{best}). It can be easily verified that these two updates are virtually identical to the HC-ACO iteration-best and rank-based updates respectively. The only difference between cGA and HC-ACO is in the form of the evaporation factor. In cGA it is equal to $\frac{\rho}{m}\sum_{s\in S} s_i$, whereas in HC-ACO it is equal to ρp_i, which is simply the expected value of the former.

4 Empirical Comparison

In this section we describe the results of the empirical comparison between the MBS algorithms described above, using MAXSAT as a test bed. MAXSAT is the optimization variant of SAT, the first problem which was shown to be NP-complete. The weighted MAXSAT problem can be formulated as follows. Given k clauses $C_1, \dots, C_k$ over n binary variables $x_1, \dots, x_n$, and the weights $w_1, \dots, w_k$, find an assignment which maximizes the sum of the weights of the satisfied clauses.

4.1 Comparison Setting

The comparison was carried out using randomly generated weighted MAXSAT instances from the SATLIB MAXSAT Benchmark Collection [10]. The benchmark set contains three groups of problems, with 100, 500 and 1000 variables and 500, 5000 and 10000 clauses respectively. Each group contains 10 instances.

The algorithms were evaluated using three different running times. For every problem size, three stopping times, T_1, T_2 and T_3, were chosen as the time that it takes for HC-ACO with population size 10 to perform 50, 200 and 1000 iterations respectively[3].

All of the algorithms described above have one or more parameters, whose choice can clearly affect the performance of the algorithm. Moreover, the optimal parameter setting may depend on the available computational time (e.g., with shorter times a higher learning rate and smaller samples should be more appropriate) and the problem size. Since to-date there are no established methods for the automatic tuning of metaheuristics' parameters, it was decided to use one problem out of every group for tuning the algorithms, and the other 9 for the testing. Specifically, each algorithm was run with a variety of parameter settings (described below), 10 times for each setting, and for every algorithm/problem-size/running-time combination, the configuration with the best average performance was chosen. This automatic tuning procedure insures that the comparison is not biased in favor of one of the algorithms. The separation between the training problem and the test problems guarantees the statistical validity of the performance estimates[4].

For all the algorithms, we considered learning rate $\rho \in \{0.03, 0.1, 0.3, 1\}$, and sample sizes $|S| \in \{10, 50, 200\}$. In HC-ACO and SGA we considered pheromone bounds with $\epsilon \in \{0, 0.01, 0.1\}$. For PBIL, the additional parameters were: "negative learning rate" $\rho_{nl} \in \{0, \rho/10, \rho\}$, mutation probability $p_{mut} \in \{0, 0.01, 0.1\}$ and mutation shift $\rho_{mut} \in \{0.01, 0.1\}$.

[3] It should be noted that, due to extensive code reuse in the algorithms' implementation, the running times for all the algorithms (without the local search) were virtually the same.

[4] If, for example, we chose the best performing configuration for *every problem* individually, the resulting average performance would no longer be an unbiased estimate of the actual expected performance.

For HC-ACO, SGA and PBIL we have tested both the iteration-best and the top-quality updates. We have also tested separately the basic CE method, which corresponds to the top-quality HC-ACO algorithm with learning rate 1 and no bounds on probabilities. Finally, both " single tournament" (cGA_{st}) and "round-robin tournament" (cGA_{rr}) versions of the cGA method were tested.

Furthermore, for every algorithm described above, we have also considered a hybrid version, in which the update was based on the population of the elite solutions, that is the highest quality solutions found so far, rather than on the last sample. Finally, all the algorithms (including the hybrid versions) were tested both with and without the use of the local search[5].

4.2 Comparison Results

Every algorithm was run 30 times on every problem with the parameter setting determined using the tuning procedure described in the previous section. Since the optimal solution costs for the benchmark problems used in this study are not known, the cost of the best solution found by *any* of the algorithms in all the test runs were used as estimates. The performance of a single run of the algorithm was evaluated as:

$$\tilde{f} = \frac{f_{best} - f_{opt}}{E\{f\} - f_{opt}}, \tag{8}$$

where f_{best} is the cost of the best solution found during the run, f_{opt} is the estimate of the optimal solution cost and $E\{f\}$ is the expected cost of the solution generated from a uniform distribution[6]. Note that, for a random solution, $E\{\tilde{f}\} = 1$, hence the results coming from different problems are put on a same scale, which allows meaningful averaging over several problems. The results of the comparison are summarized in Tables 1 and 2. Every column corresponds to a comparison with a particular problem size and stopping time. The average score of the empirically best algorithm is printed in bold typeface and the results, which are worse than the best one with 95% confidence[7], are shown in italic. Since the use of the local search leads to a drastic improvement of performance, the results in Table 2 are multiplied by 100 for clarity of presentation. The superscript "h" denotes the hybrid versions of the algorithms, augmented with population, and "n" denotes the problem size, measured by the number of variables.

When local search is not used, in most cases the "round-robin" tournament cGA produces significantly better results. Still, even the performance of cGA

[5] Although using local search is not a common practice in the EDA research field, the results reported next indicate that it certainly should be considered in the future.

[6] Since, for any clause C with d variables, the proportion of non-satisfying assignments is $1/2^d$, it can be verified that, for the 3-MAXSAT problems used in the benchmarks, $E\{f\} = \frac{7}{8}\sum_{j=1}^{k} w_j$.

[7] The statistical analysis was performed using Tukey-Kramer test, which is a modification of the t-test, adapted for the multiple comparisons.

Table 1. Average performance of the algorithm without the local search.

	n=100			n=500			n=1000		
	T_1	T_2	T_3	T_1	T_2	T_3	T_1	T_2	T_3
CE	*0.2561*	*0.1294*	*0.1033*	*0.5567*	*0.4870*	*0.2026*	*0.6642*	*0.5795*	*0.2585*
CE^h	*0.2541*	*0.1497*	*0.1021*	*0.5646*	*0.4854*	*0.1993*	*0.6676*	*0.5784*	*0.2586*
$HC\text{-}ACO_{top}$	*0.2039*	*0.1230*	*0.0778*	*0.5567*	*0.2979*	*0.1547*	*0.6439*	*0.4251*	*0.2234*
$HC\text{-}ACO^h_{top}$	**0.1843**	0.1050	*0.0714*	*0.5138*	*0.3094*	*0.1554*	*0.6301*	*0.4288*	*0.2387*
$HC\text{-}ACO_{best}$	*0.2098*	*0.1336*	*0.0762*	*0.5676*	*0.3358*	*0.1670*	*0.6695*	*0.4636*	*0.2503*
$HC\text{-}ACO^h_{best}$	0.1980	*0.1292*	*0.1063*	*0.5558*	*0.3433*	*0.1840*	*0.6719*	*0.5020*	*0.3625*
SGA_{top}	*0.2613*	*0.1370*	*0.0879*	*0.5398*	*0.3160*	*0.1588*	*0.6417*	*0.4184*	*0.2153*
SGA^h_{top}	*0.2349*	*0.1330*	*0.0773*	*0.5397*	*0.3139*	*0.1476*	*0.6475*	*0.4268*	*0.2101*
SGA_{best}	*0.2692*	*0.1437*	*0.0808*	*0.5804*	*0.3473*	*0.1625*	*0.6854*	*0.4556*	*0.2242*
SGA^h_{best}	*0.2709*	*0.1409*	*0.1065*	*0.6005*	*0.3694*	*0.1640*	*0.7049*	*0.4973*	*0.2350*
$PBIL_{top}$	*0.2311*	*0.1545*	**0.0464**	*0.5058*	*0.4015*	*0.1549*	*0.6063*	*0.4816*	*0.2266*
$PBIL^h_{top}$	*0.2868*	*0.1790*	*0.1038*	*0.6613*	*0.3991*	*0.2133*	*0.7300*	*0.4731*	*0.2491*
$PBIL_{best}$	*0.2837*	*0.1206*	0.0504	*0.5658*	*0.3072*	*0.1415*	*0.6948*	*0.4380*	*0.2554*
$PBIL^h_{best}$	*0.3214*	*0.1925*	*0.1094*	*0.7399*	*0.4458*	*0.1576*	*0.7900*	*0.5169*	*0.2635*
cGA_{rr}	0.1949	**0.0979**	*0.0746*	**0.4168**	**0.2606**	**0.1086**	**0.5395**	**0.3947**	**0.1528**
cGA^h_{rr}	*0.4625*	*0.2244*	*0.1339*	*0.7500*	*0.6292*	*0.4225*	*0.8163*	*0.7252*	*0.5229*
cGA_{st}	*0.3007*	*0.1455*	*0.0814*	*0.5891*	*0.3391*	*0.1535*	*0.6816*	*0.4561*	*0.2498*
cGA^h_{st}	*0.4627*	*0.3715*	*0.3307*	*0.7687*	*0.7070*	*0.6485*	*0.8351*	*0.7862*	*0.7311*

is relatively poor (recall that the expected performance score of a randomly generated solution is 1). The use of local search leads to an improvement of almost two orders of magnitude and there seems to be almost no significant differences between the algorithms in this case (note, however, that cGA is often significantly worse than the others). We hypothesize that the differences among the algorithms with the local search could be just an artifact of the particular tuning procedure which we use, rather than an evidence of the advantage of one of the methods. This is the topic of ongoing research.

5 Conclusions

During the last decade a new approach for solving combinatorial optimization problems has been emerging, as already observed in [13]. This approach, which we refer to as model-based search (MBS), tackles the combinatorial optimization problem by sampling the solution space using a probabilistic model, which is adaptively modified as the search proceeds. In this paper we presented a comparative analysis of several existing MBS methods, which construct binary coded solutions by generating every bit independently. Our theoretical analysis revealed considerable structural similarity among these algorithms, and the empirical comparison showed that also the actual performance of the algorithms is quite similar (especially, when the algorithms are hybridized with local search). In the future we hope to extend our analysis to a more general class of MBS algorithms and to compare these algorithms on different types of combinatorial optimization problems.

Table 2. Average performance (multiplied by 100) of the algorithm with the local search.

	n=100			n=500			n=1000		
	T_1	T_2	T_3	T_1	T_2	T_3	T_1	T_2	T_3
CE	0.1816	*0.0300*	**0.0000**	**0.4199**	*0.2416*	*0.1173*	0.8352	*0.4494*	*0.2125*
CE^h	0.2296	*0.0216*	**0.0000**	0.4264	0.2029	*0.1460*	0.7944	*0.4586*	0.1536
$HC\text{-}ACO_{top}$	0.1809	0.0057	0.0016	0.4810	0.2055	0.0817	0.8352	0.3266	0.1203
$HC\text{-}ACO^h_{top}$	0.2331	0.0079	0.0023	0.5066	0.2124	0.0895	0.8440	*0.4586*	**0.0857**
$HC\text{-}ACO_{best}$	0.1303	0.0078	0.0023	0.4846	0.2207	0.0728	*0.9838*	0.3876	*0.1811*
$HC\text{-}ACO^h_{best}$	0.1536	0.0096	*0.0070*	0.4951	0.2160	0.0713	*0.9884*	*0.4991*	0.1324
SGA_{top}	**0.1233**	0.0067	**0.0000**	0.4324	0.2094	0.0823	0.8354	0.3491	0.1247
SGA^h_{top}	0.1934	0.0048	**0.0000**	0.4298	**0.1548**	**0.0485**	0.8230	0.3433	0.1194
SGA_{best}	0.1573	0.0039	0.0054	*0.6292*	*0.2404*	0.0634	0.8929	*0.5087*	0.1416
SGA^h_{best}	0.2104	0.0189	0.0031	*0.6242*	0.2194	*0.1015*	*0.9078*	0.3883	0.1425
$PBIL_{top}$	0.1586	**0.0017**	0.0024	0.4844	0.1899	*0.1107*	0.8353	**0.2925**	0.1113
$PBIL^h_{top}$	0.1729	0.0054	0.0008	0.4472	0.1973	*0.1203*	**0.7821**	0.3026	0.1124
$PBIL_{best}$	0.1606	0.0110	*0.0066*	0.4292	0.1771	0.0673	*0.9078*	*0.4275*	0.1288
$PBIL^h_{best}$	0.1392	0.0192	0.0023	0.4734	*0.2313*	*0.1249*	*0.9074*	*0.4095*	0.1537
cGA_{rr}	0.1737	0.0137	**0.0000**	*0.6502*	*0.2937*	0.0792	*1.3845*	*0.5153*	0.1508
cGA^h_{rr}	0.1580	0.0086	**0.0000**	*0.6792*	0.2269	0.0503	*1.4152*	*0.6443*	*0.1931*
cGA_{st}	0.1600	*0.0257*	0.0047	*0.6212*	*0.2673*	*0.1198*	*0.9264*	*0.4903*	*0.2114*
cGA^h_{st}	0.1568	*0.0248*	0.0031	*0.6713*	*0.2953*	*0.1368*	*0.9054*	*0.6220*	*0.2843*

Acknowledgments

Mark Zlochin is supported through a Training Site fellowship funded by the Improving Human Potential (IHP) programme of the Commission of the European Community (CEC), grant HPRN-CT-2000-00032. Marco Dorigo acknowledges support from the Belgian FNRS, of which he is a Senior Research Associate. More generally, this work was partially supported by the "Metaheuristics Network", a Research Training Network funded by the Improving Human Potential programme of the CEC, grant HPRN-CT-1999-00106. The information provided in this paper is the sole responsibility of the authors and does not reflect the Community's opinion. The Community is not responsible for any use that might be made of data appearing in this publication.

References

1. S. Baluja and R. Caruana. Removing the genetics from the standard genetic algorithm. In *Proceedings of ICML'95*, pages 38–46. Morgan Kaufmann Publishers, Palo Alto, CA, 1995.
2. C. Blum, A. Roli, and M. Dorigo. HC–ACO: The hyper-cube framework for Ant Colony Optimization. In *Proceedings of MIC'2001*, volume 2, pages 399–403, Porto, Portugal, 2001.
3. J. S. de Bonet, C. L. Isbell, and P. Viola. MIMIC: Finding optima by estimating probability densities. In *Proceedings of NIPS'97*, pages 424–431. MIT Press, Cambridge, MA, 1997.

4. G. Di Caro and M. Dorigo. AntNet: Distributed stigmergetic control for communications networks. *Journal of Artificial Intelligence Research*, 9:317–365, 1998.
5. M. Dorigo. *Ottimizzazione, Apprendimento Automatico ed Algoritmi Basati su Metafora Naturale*. PhD thesis, Dipartimento di Elettronica, Politecnico di Milano, Milan, Italy, 1992.
6. M. Dorigo and G. Di Caro. The Ant Colony Optimization meta-heuristic. In D. Corne, M. Dorigo, and F. Glover, editors, *New Ideas in Optimization*, pages 11–32. McGraw Hill, London, UK, 1999.
7. M. Dorigo and L. M. Gambardella. Ant Colony System: A cooperative learning approach to the traveling salesman problem. *IEEE Trans. on Evol. Comp.*, 1(1):53–66, 1997.
8. M. Dorigo, M. Zlochin, N. Meuleau, and M. Birattari. Updating ACO pheromones using Stochastic Gradient Ascent and Cross-Entropy methods. In *Proceedings of EvoWorkshops 2002*, pages 21–30. Springer Verlag, Berlin, Germany, 2002.
9. G. R. Harik, F. G. Lobo, and D. E. Goldberg. The compact genetic algorithm. *IEEE Trans. on Evol. Comp.*, 3(4):287–297, 1999.
10. H. H. Hoos and T. Stützle. Randomly generated benchmark problems for MAXSAT. Technical Note, Department of Computer Science, University of British Columbia, March 2001.
11. P. Larrañaga and J.A. Lozano. *Estimation of Distribution Algorithms. A New Tool for Evolutionary Computation.* Kluwer Academic Publishers, 2001.
12. N. Meuleau and M. Dorigo. Ant colony optimization and stochastic gradient descent. *Artificial Life*, 8(2):103–121, 2002.
13. N. Monmarché, E. Ramat, G. Dromel, M. Slimane, and G. Venturini. On the similarities between AS, BSC and PBIL: toward the birth of a new meta-heuristic. Technical Report 215, Laboratoire d'Informatique, Université de Tours, 1999.
14. M. Pelikan, D. E. Goldberg, and E. Cantú-Paz. BOA: The Bayesian optimization algorithm. In *Proceedings of GECCO'99*, volume I, pages 525–532. Morgan Kaufmann Publishers, San Francisco, CA, 1999.
15. J. Quinlan. Combining instance-based and model-based learning. In *Proceedings of the Twelfth International Conference on Machine Learning (ML-93)*, pages 236–243. Morgan Kaufmann Publishers, San Mateo, CA, 1993.
16. H. Robbins and S. Monro. A stochastic approximation method. *Annals of Mathematical Statistics*, 22:400–407, 1951.
17. A. Roli, C. Blum, and M. Dorigo. ACO for maximal constraint satisfaction problems. In *Proceedings of MIC'2001*, volume 1, pages 187–191, Porto – Portugal, 2001.
18. R. Y. Rubinstein. The cross-entropy method for combinatorial and continuous optimization. *Methodology and Computing in Applied Probability*, 1(2):127–190, 1999.
19. T. Stützle and H. H. Hoos. $\mathcal{MAX}$–$\mathcal{MIN}$ Ant System. *Future Generation Computer Systems*, 16(8):889–914, 2000.
20. M. Zlochin, M. Birattari, N. Meuleau, and M. Dorigo. Model-based search for combinatorial optimization. Technical Report TR/IRIDIA/2001-15, IRIDIA, Université Libre de Bruxelles, 2001.