A simulated annealing method based on a specialised evolutionary algorithm

C. García-Martínez \textsuperscript{a,\ast}, M. Lozano \textsuperscript{b}, F.J. Rodríguez-Díaz \textsuperscript{b}

\textsuperscript{a} Department of Computing and Numerical Analysis, University of Córdoba, 14071, Spain
\textsuperscript{b} Department of Computer Science and Artificial Intelligence, CITIC-UGR (Research Center on Information and Communications Technology), University of Granada, 18071, Spain

1. Introduction

Metaheuristics (MHs)\cite{1,2} arose around 30 years ago, extending basic heuristic methods by means of their application in iterative frameworks that augmented their exploration capabilities. Blum and Roli\cite{3} offer an overview of various existing methods. These stochastic algorithms have proved to be useful when coping with difficult optimisation problems because they usually obtain good solutions in a reduced amount of time.

Over the last years, a large number of search algorithms were reported that do not purely follow the concepts of one single classical MH, but they attempt to obtain the best from a set of MHs (and even other kinds of optimisation methods) that perform together and complement each other to produce a profitable synergy from their combination. These approaches are commonly referred to as hybrid MHs\cite{4–6}.

Simulated annealing (SA)\cite{7,8} is commonly said to be the first algorithm extending local search methods with an explicit strategy to escape from local optima. The fundamental idea is to allow moves resulting in solutions of worse quality than the current solution in order to escape from local optima. The probability of doing such a move is decreased during the search process. Despite of being proposed in 1983, SA is still object of further studies, tool tackling many optimisation problems, and component of other search algorithms\cite{9–14}. Precisely, its outstanding role in the MH field encourages further studies to obtain more effective SA models.

Evolutionary algorithms (EAs)\cite{15,16} rely on the concept of a population of individuals, which undergo probabilistic operators such as mutation, selection, and (sometimes) recombination to evolve toward increasingly better fitness values of the individuals. Genetic Algorithms, Evolutionary Strategies, Evolutionary Programming and Genetic Programming are examples of this family of MHs.

The flexibility offered by the evolutionary paradigm allows specialised models to be obtained with the aim of performing as other search methods do, but more satisfactorily. This practice is a recent alternative to design new hybrid MHs, and an overview of specialised EAs on intensification and diversification can be found in\cite{17}. In particular, there exist several evolutionary proposals acting as local search methods and performing more effectively\cite{18–21}.

The design of hybrid MHs with ideas from the SA and EAs fields is a fruitful research line. There exist several proposals in the literature that either introduce the acceptance mechanism of SA in an EA\cite{22,23}, or apply a SA algorithm to optimise the members of the population of an EA\cite{24}, or exploit the advantage of a population of SA processes\cite{25,26}. In this paper, we present SA based on a specialised EA (SASEA), an innovative hybrid MH that brings out an alternative scheme for combining ideas from SA and EAs. SASEA can be seen as either a specialised EA that carries out the search process typical of SA, or a SA approach based on ideas of the EA field. It consists in a steady-state EA\cite{27,28} that creates one single candidate solution at each iteration, by crossing over a given solution, or current solution, and another one from the population. Afterwards, SASEA applies an acceptance mechanism to decide which solution

\textsuperscript{\ast} Corresponding author. Tel.: +34 957 212660; fax: +34 957 218630. E-mail address: cgarcia@uco.es (C. García-Martínez).

1568-4946/ - see front matter © 2011 Elsevier B.V. All rights reserved.
doi:10.1016/j.asoc.2011.11.007
becomes the new current solution, either the candidate solution or the current one. The other solution is inserted into the population by a replacement strategy.

We have carried out an experimental study, deeply extending the one presented in García-Martínez and Lozano [29], on a test suite composed of binary combinatorial optimisation problems, with the aim of analysing the benefits of SASEA with regard to other SA approaches, hybrid MHS with ideas from SA and EAs, and other state-of-the-art search algorithms for this kind of problems. We have concluded that the innovative scheme for combining ideas from SA and EAs introduced by SASEA may improve the performance of this kind of hybrid MHS, and achieve competitive results with regard to state-of-the-art search algorithms.

The paper is set up as follows. In Section 2, SA is introduced. In Section 3, we present SASEA. Its components are deeply described. In Section 4, we define the empirical framework used to analyse the benefits of SASEA. In Section 5, we analyse the neighbourhood structure introduced by SASEA with regard to related work presented in the literature. We also conduct an empirical comparison with two SA proposals and CHC. In Section 6, we outline the differences between SASEA and other hybrid MHS combining ideas from EAs and SA. The benefits of SASEA with regard to those hybrid MHS are empirically studied. In Section 7, we pit SASEA against state-of-the-art search algorithms for binary-coded problems. Finally, conclusions and future works are presented in Section 8.

2. Simulated annealing

SA [8,7] is an optimisation technique analogous to the physical process of annealing. SA starts with a high temperature $T$ and any initial state $X^\circ$. A neighbourhood operator is applied to the current state $X^\circ$ (having energy $f(X^\circ)$) to yield state $Z$ (having energy $f(Z)$) (see Fig. 1). When tackling binary combinatorial optimisation problems, the most frequently applied neighbourhood operator is the one-flip, which flips one bit of the current solution. Then, an acceptance mechanism decides which state becomes the new current state. Acceptance mechanism takes into account the current temperature of the system: worse solutions are more often accepted with high temperature. The application of the neighbourhood operator and the probabilistic acceptance of newly generated states are repeated either a fixed number of iterations or until a quasi-equilibrium is reached. The entire above-described procedure is performed repeatedly, each time starting from the current state and from a lower $T$.

The idea behind SA is to protect diversification in the initial stages and intensification later: at the initial stages, high $T$ values favour the exploration of the search space, accepting new states almost regardless their energy; later, low $T$ values increase exploitation, accepting only new better states.

There are two main acceptance mechanisms proposed in the literature: metropolis and logistic rules [30]. They define the probability of $Z$ being the new current state. Eqs. (1) and (2) show metropolis and logistic acceptance mechanisms, respectively, for minimisation problems. On the other hand, there exist several cooling schemes, such as logarithmic, fast, and geometric ones [30]. Eq. (3) shows geometric cooling scheme, which is used in this work ($\alpha$ is a cooling factor often assumed to be a constant in the interval $[0.9, 1]$):

\[ p(Z) = \begin{cases} 
1 & \text{if } f(Z) < f(X^\circ) \\
\frac{1}{1 + e^{(f(X^\circ) - f(Z))/\beta}} & \text{if } f(Z) \geq f(X^\circ) 
\end{cases} \]  

\[ p(Z) = 1 - \frac{1}{1 + e^{(f(X^\circ) - f(Z))/\beta}} \]  

\[ T \leftarrow \alpha \cdot T \]  

Convergence to global optimum is guaranteed for SA if a stationary distribution is reached at each temperature, followed by sufficiently slow cooling, such as logarithmic [31]. However, it usually leads to excessive long runs and practitioners use to apply faster cooling schemes at fixed number of iterations [32].

3. SA based on a specialised EA

SASEA is a steady-state EA that creates one single candidate solution at each iteration, by crossing over the current solution and another one from the population. Afterwards, SASEA applies an acceptance mechanism, e.g. metropolis or logistic, to decide which solution becomes the new current solution, either the candidate solution or the current one. The other solution is inserted into the population by a replacement strategy.

An outstanding feature of SASEA is that it describes a trajectory in the search space, as classical SA procedures do. Both, classical SA algorithms and SASEA, commence from a single solution and, at each step, a candidate solution is generated using a neighbourhood operator of some sort. They simply move the search from the current solution to a candidate solution according to the acceptance mechanism. The basic idea of SASEA is to use a SA accepting mechanism as the move accepting criterion of the search and crossover as the neighbourhood operator. In fact, the neighbourhood structure defined by the crossover operator of SASEA, together with the individual selected from the population, is one of the most important differences with regard to classic SA methods. Several researchers have pointed out that crossover operators may promote neighbourhood structures [33–36]. Precisely, first trajectory methods based on crossover were suggested in 1995 [37,38], and they have been applied to obtain different local EAs [20,21]. The main novelty of SASEA concerns the application of these ideas to design a new SA model.

SASEA can be seen as an extension of Binary-coded Local Genetic Algorithm (BLGA) [39,19], a recent EA that performs local search to an external solution. BLGA was compared with classic local search methods obtaining promising results with regard to efficacy and efficiency. SASEA extends BLGA by including the concepts that distinguish SA from local search methods: temperature, acceptance criterion, and cooling scheme. In addition, some components are modified in order to better fit the SA search process.

In Section 3.1, we describe steady-state EAs. In Section 3.2, we introduce the general scheme of SASEA. Sections 3.3, 3.4 and 3.5 describe in detail, selection mechanism, crossover operator and replacement strategy of SASEA, respectively.
3.1. Steady-state EAs

The generational EA creates new offspring from the members of an old population using the genetic operators and places these individuals in a new population that becomes the old population when the whole new population is created. The steady-state EA [27,28] is different to the generational model in that there is typically one single new member inserted into the new population at any one time. A replacement/deletion strategy defines which member in the current population is forced to perish (or vacate a slot) in order to make room for the new offspring to compete (or, occupy a slot) in the next iteration. Steady-state EAs are overlapping systems, since parents and offspring compete for survival.

The basic algorithm step of steady-state EAs is shown in Fig. 2. These steps are repeated until a termination condition is achieved. In step 4, one can choose the replacement strategy (e.g., replacement of the worst, the oldest, or a randomly chosen individual). In step 5, one can choose the replacement condition (e.g., replacement if the new individual is better or unconditional replacement).

3.2. General scheme of SASEA

Fig. 3 outlines the pseudocode of SASEA. It starts with a high temperature $T$, an initial state $X^c$, and a randomly uniformly generated population ($Pop$). Then, the following steps are carried out (see Fig. 4):

1. **Mate selection**: an individual $Y$ is selected from the population applying parametrised assortative mating (Section 3.3).
2. **Crossover**: $X^c$ is crossed over with $Y$ by means of guiding neighbourhood exploration crossover operator, generating offspring $Z$ (Section 3.4). Notice that the crossover-based neighbourhood structure has been depicted as a cloud in Fig. 4 to stress the difference with classic neighbourhood structures of SA.
3. **Acceptance**: $Z$ replaces $X^c$ according to the applied acceptance mechanism (metropolis, logistic, etc.).

![Fig. 2. Basic scheme of steady-state EAs.](image)

![Fig. 4. SASEA model.](image)

4. **Update population**: if $Z$ replaced $X^c$, then old $X^c$ is inserted into the population using restricted tournament selection (Section 3.5). Otherwise, $Z$ is inserted in the population using the same replacement scheme. This replacement strategy helps SASEA to maintain a diverse set of solutions in $Pop$.
5. **Annealing**: if the cooling condition has been reached, apply the selected cooling scheme, e.g., geometric cooling scheme.

All these steps are repeated until termination condition is reached.

3.3. Parametrised assortative mating

Assortative mating is the natural occurrence of mating between individuals of similar phenotype more or less often than expected by chance. Mating between individuals with similar phenotype more often is called positive assortative mating and less often is called negative assortative mating. Fernandes and Rosa [40,41] implement these ideas to design two mating selection mechanisms. A first parent is selected by the roulette wheel method and $n_{ma}$ individuals are selected with the same method. Afterwards, similarity between each of these individuals and first parent is computed (similarity between two binary-coded solutions is defined as the Hamming distance between them). If assortative mating is negative, then the one with less similarity is chosen. If it is positive, the genome more similar to the first parent is chosen to be the second parent.

We introduce a new mating mechanism, called parametrised assortative mating, that probabilistically regulates similarity between mated individuals. First parent is always $X^c$ and second parent is chosen being more or less similar to $X^c$ according to a parameter of the operator ($p$) (ties are broken up by selecting one element uniformly at random). Pseudocode of parametrised assortative mating is presented in Fig. 5 ($Random(Pop)$, uniformly samples a solution from $Pop$).

Parameter $p$ adjusts likeness mating trends of $X^c$. This parameter takes values in the interval $[0,1]$. If $p=0$, $X^c$ tends to mate dissimilar individuals and $p$ moves closer to 1, $X^c$ tends to mate individuals more alike.

With the aim of preserving SA idea, to protect diversification in the initial stages and intensification later, SASEA applies a deterministic adaptation rule [42] on parameter $p$. At each iteration, it is set to the fraction of consumed evaluations:

$$p = \frac{FEs}{MaxFEs},$$  \tag{4}$$

where $FEs$ is the number of consumed evaluations and $MaxFEs$ is the maximum number of evaluations allowed. The objective is to mate dissimilar individuals at the initial stages of the search.
process, when $p$ is close to 0, favouring diversification. Then, to gradually tend to mating similar individuals at latter stages, because $p$ linearly approaches value 1, increasing intensification.

### 3.4. Guiding neighbourhood exploration crossover operator

SASEA applies a specific crossover, called **guiding neighbourhood exploration crossover operator**, that is aimed at guiding the exploration of the neighbourhood of $X^c$. It carries through two stages. At the first stage, offspring is pushed from $X^c$ toward the mate $Y$. $Y$ can be seen as an agent guiding the exploration of $X^c$-neighbourhood. $Y$ also defines the intensity by which offspring is pushed away. If $Y$ is close to $X^c$, offspring is sampled next to $X^c$, otherwise, if $Y$ is far from $X^c$, offspring undergoes a stronger drive. At the second stage, offspring experiences a stochastic perturbation depending on the intensity of previous drive. The aim of the second stage is to add randomness to neighbourhood exploration. This perturbation may reduce, enforce previous drive, or even add new genetic material. Fig. 6 represents the operation of the crossover operator.

Fig. 7 outlines pseudocode of guiding neighbourhood exploration crossover operator. Random returns one element from the given set, uniformly sampled at random, and $N$ refers to the number of bits of the problem. Given $X^c$ and a mate $Y$:

1. The operator initially creates offspring $Z$ as a copy of $X^c$ and detects differences with $Y$.

![Guiding neighbourhood exploration crossover operator](Image)

**Fig. 6.** Guiding neighbourhood exploration crossover operator.

2. First stage: genes from $Y$, where $Y$ and $X^c$ differ, are included in $Z$ with probability $p_Y$, parameter of the operator.

3. Second stage: genes are flipped according to a probability that depends on the number of genes from $Y$ included in previous stage.

4. At last, if offspring $Z$ has not included genes from $Y$ nor experienced perturbation, a randomly uniformly chosen gene is flipped.

It is interesting to mention that this new crossover operator shares some similarities with **half uniform crossover operator** (HUX) of CHC [43]. In particular, first stage of guiding neighbourhood and half uniform crossover operators actuate according to the differences detected between both parents. Whereas half uniform operator assigns exactly half of these differences to each produced

**Fig. 7.** Pseudocode of guiding neighbourhood exploration crossover operator.
Input: Pop, I, n_T
Output: Pop

G_T ← Select n_T random members from Pop;
R ← Find member from G_T most similar to I;
if I is better than R then
   Pop ← (Pop \ R) ∪ I;
end

Fig. 8. Restricted tournament selection.

offspring, guiding neighbourhood exploration crossover operator considers control parameter p_T for that assignment.

3.5. Restricted tournament selection

Crowding methods [44,45] attempt to maintain diversity in the population by means of the replacement procedure as follows: new individuals are more likely to replace existing individuals in the parent population that are similar to themselves based on genotypic similarity. They have been used for locating and preserving multiple local optima in multimodal functions.

SASEA uses restricted tournament selection [46] as the replacement method. This strategy was applied previously by BGAL [39,19]. Its main idea is to replace the closest individual R to the one being inserted in the population, I, from a set of n_T randomly selected ones, uniformly and with-replacement, if I is better than R. Possible ties are broken up by choosing one individual uniformly at random. Fig. 8 outlines its pseudocode.

4. Empirical framework

We have carried out different experiments to analyse and assess the efficacy of SASEA with regard to other algorithms for binary-coded problems. In this section, we detail the test problems (Section 4.1) and running conditions (Section 4.2) that were used for the following studies. Some comments on the no free lunch theorems [47] are addressed in Section 4.3.

4.1. Test problems

Experiments were executed on a test suite composed of 27 binary-coded test problem instances from 12 different problem classes, 8 classes (13 instances) from the artificial intelligence field and 4 (14 instances) from real-world applications. They are described in Appendix A. Table 1 outlines their name, number of bits, and a value (f) that stands for either the fitness value of the global optimum, known best solution, or upper bound presented in the literature. All of them have been formulated as maximisation problems. BQP and Maxcut instances can be obtained from the corresponding files from the Big Mac Library,1 and Multiple knapsack problems, from the SAC-94 Suite.2 Though Table 1 indicates that 1 is the maximum possible fitness value for M-Sat and NkLand problems, there not usually exists any optimal solution with that fitness value, which depends on the current problem instance.

For problems where the global optimum is the all-ones string (Royal-road, Trap, Deceptive, Bipolar deceptive, Overlapping deceptive, and HIFF ones), we firstly build a random bit string S that will become the shifted optimum of the problem. The new fitness value of a candidate solution X is computed as the original fitness value of the string X ⊕ S (i.e., the result of applying first exclusive OR between X and S, and subsequently, inversion). This way, we avoid the biased advantages of some search algorithms that tend to sample the all-ones string at the beginning of the search process.

4.2. Running conditions

One advantage of trajectory MHs over other heuristics is that, when solving some problems, the search space can be searched very efficiently: instead of calculating the objective value of a new candidate solution, it is sufficient to compute the difference ∆f with regard to the fitness of the current solution by utilising problem-specific knowledge. This technique, widely known as delta evaluation [48], is highly profitable in terms of computation time whenever feasible. For example, in the BQP problem, the calculation of ∆f takes time O(n), while the calculation of the fitness takes O(n^2).

As delta evaluation can be performed on several of the tackled problems (and many others [49]), it will be applied by all the trajectory search methods (or trajectory search components) considered. In particular, SASEA will apply delta evaluation as well, which is possible due to guiding neighbourhood exploration crossover operator, applying a low p_T value, creates offspring near the first parent, i.e., few bits are changed.

In most real-world optimisation problems the evaluation of a solution requires a simulation process that is usually very time-consuming, i.e., solution evaluation is the bottleneck of the search process. For that reason, in order to perform a fair comparison between different search methods, we will run every algorithm with the same budget of fitness evaluations (independently delta evaluation is being applied or not, which does not affect the algorithms’ performance in terms of efficacy). In general, each run of a search algorithm on a test problem will perform at most 10^5 fitness evaluations, but convergence characteristics will be studied as well. The performance measure is the average of the best fitness value found over 50 independent runs, because we are interested in the regular performance of the compared algorithms, instead of the best results attained.

<table>
<thead>
<tr>
<th>Prob.</th>
<th>Name</th>
<th>N</th>
<th>f</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Royal road problem (400, 8)</td>
<td>400</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>Trap problem</td>
<td>36</td>
<td>220</td>
</tr>
<tr>
<td>3</td>
<td>Deceptive problem</td>
<td>39</td>
<td>390</td>
</tr>
<tr>
<td>4</td>
<td>Bipolar deceptive problem</td>
<td>396</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>Overlapping deceptive problem</td>
<td>399</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>NkLand(48,4)</td>
<td>48</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>NkLand(48,12)</td>
<td>48</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>HIFF(2, 5, true)</td>
<td>32</td>
<td>192</td>
</tr>
<tr>
<td>9</td>
<td>HIFF(3, 4, false)</td>
<td>81</td>
<td>211</td>
</tr>
<tr>
<td>10</td>
<td>PPeaks(10,100)</td>
<td>100</td>
<td>1</td>
</tr>
<tr>
<td>11</td>
<td>PPeaks(100,100)</td>
<td>100</td>
<td>1</td>
</tr>
<tr>
<td>12</td>
<td>PPeaks(50,150)</td>
<td>150</td>
<td>1</td>
</tr>
<tr>
<td>13</td>
<td>PPeaks(50,200)</td>
<td>200</td>
<td>1</td>
</tr>
<tr>
<td>14</td>
<td>M-Sat(100,1200,3)</td>
<td>100</td>
<td>1</td>
</tr>
<tr>
<td>15</td>
<td>M-Sat(100,2400,3)</td>
<td>100</td>
<td>1</td>
</tr>
<tr>
<td>16</td>
<td>BQP(bqp50-1)</td>
<td>50</td>
<td>5176</td>
</tr>
<tr>
<td>17</td>
<td>BQP(bqp500-1)</td>
<td>500</td>
<td>121,270</td>
</tr>
<tr>
<td>18</td>
<td>BQP(be120.3.3)</td>
<td>120</td>
<td>Not known</td>
</tr>
<tr>
<td>19</td>
<td>BQP(be200.8.5)</td>
<td>200</td>
<td>Not known</td>
</tr>
<tr>
<td>20</td>
<td>Maxcut(pml1s,80.6)</td>
<td>80</td>
<td>73</td>
</tr>
<tr>
<td>21</td>
<td>Maxcut(w05_100.2)</td>
<td>100</td>
<td>2738</td>
</tr>
<tr>
<td>22</td>
<td>Maxcut(g05_100.5)</td>
<td>100</td>
<td>1436</td>
</tr>
<tr>
<td>23</td>
<td>Maxcut(pwo5_100.6)</td>
<td>100</td>
<td>8217</td>
</tr>
<tr>
<td>24</td>
<td>Maxcut(ising2.5-250,5555)</td>
<td>250</td>
<td>7,919,449</td>
</tr>
<tr>
<td>25</td>
<td>Multiple knapsack p. (weish03)</td>
<td>30</td>
<td>4115</td>
</tr>
<tr>
<td>26</td>
<td>Multiple knapsack p. (pet5)</td>
<td>28</td>
<td>12,400</td>
</tr>
<tr>
<td>27</td>
<td>Multiple knapsack p. (pb4)</td>
<td>29</td>
<td>95,168</td>
</tr>
</tbody>
</table>

1 http://biqmac.uni-kl.de/.
2 http://elib.zib.de/pub/Packages/mp-testdata/ip/sac94-suite/.
Non-parametric tests can be used for comparing the results of different optimisers [50–52]. Given that the non-parametric tests do not require explicit conditions for being conducted, it is recommended that the sample of results are obtained following the same criterion, which is, to compute the same aggregation (average, mode, etc.) over the same number of runs for each search algorithm and problem. In particular, we have used the Wilcoxon matched-pairs signed-ranks [53] test to compare the results of our proposal with the ones of other MHS. We explain this statistical test, in detail, in Appendix B.

4.3. On the no free lunch theorem

No free lunch theorem for optimisation [47] states that all not-resampling algorithms that search for an extremum of a cost function in a finite space perform exactly the same when averaged over all possible fitness functions.

Since, the application of not-resampling algorithms on problems with many variables usually involves impossible memory and/or computation requirements (a general binary-coded problem with \(N\) variables requires \(2^N\) evaluations to be solved to optimality), resampling is allowed for most MHS. Though, no free lunch theorem is not applicable to those cases, there is a belief, as many researchers write so, that all (resampling) MHS might perform exactly the same when averaged over all possible fitness functions, too. In particular, proposals’ deficiencies are often explained by sentences like “According to the no free lunch theory, it is impossible for algorithm X to outperform always”, even when they are being proposed. This fact points out the significance of this theorem. Recently, Marshall and Hinton [54] have proved that NFL theorems do not hold when considering resampling algorithms, however, random search with replacement is expected anyway to be the best approach when performances are averaged over all possible functions (or closed under permutation sets, as defined below).

Schumacher et al. [55] revealed that no free lunch theorem even holds when averaging over subclasses of problems if and only if these sub-classes are closed under permutations if for any function \(f \in F\) and any permutation \(\pi\) of the search space, \(f \circ \pi\) is also in \(F\) (notice that the number of permutations of a binary-coded search space with \(N\) variables is \(2^N!\)). Later, Igel and Toussaint [56] suggested that the class of real-world applications is hardly closed under permutations. In particular, they proved that the class of problems with some topological structures based on nontrivial neighbourhood relations is not closed under permutations. This result implies that it might be possible to detect performance differences between some algorithms when averaged over the class of all real-world problems.

Since, our tested is not closed under permutations, because there exist some neighbourhood relations between the solutions in their search spaces (notice that the procedure for shifting the optimum of several problems does not close them under permutations because neighbourhood relations are not affected and, just because a maximum of \(2^N\) transformations are possible), it is expected that the compared algorithms might not perform exactly the same on average, and we could rank them according to the average performance. Our intention is to analyse whether SASEA is able to outperform its competitors when averaged on the considered testbed and under the specified running conditions, not to defy the no free lunch theorem, but to forecast similar performances on other real-world problems (with similar neighbourhood relations) with regard to the compared algorithms. On the contrary, we know that the set containing all the binary static combinatorial problems is really closed under permutations, and thus, any averaged performance difference should not to be expected. Finally, we shall mention that, the practice followed in this work is widely applied by the evolutionary computation community [57,58,34,59–63].

5. Study of the neighbourhood structure of SASEA

The neighbourhood structure of SASEA, as the result of the combination of guiding neighbourhhood exploration crossover operator and the way parametrised assortative mating selects individuals from the population, is a major difference with classic SA methods. In Section 5.1, we point out the properties of the neighbourhood structure promoted by SASEA. In Section 5.2, we empirically study whether the new neighbourhood structure of SASEA outperforms other SA neighbourhood structures for binary-coded problems proposed in the literature. Besides, we empirically analyse the performance synergy between guiding neighbourhhood exploration crossover operator and SASEA with regard to the one between HUX and CHC in Section 5.3, according to the proper adjustment of its parameter \(p_x\).

5.1. Properties of the neighbourhood structure of SASEA

The neighbourhood structure promoted by SASEA presents the following two properties:

- **It explores larger and dynamic neighbourhood sizes:** classic SA algorithms for binary optimisation usually consider one-flip operator, which flips just one bit of the current solution, to explore the neighbourhood of \(X^*\). However, SASEA may flip several bits of \(X^*\) and the number of flips is adapted dynamically at each iteration. When \(X^*\) is crossed over with dissimilar individuals, offspring undergoes strong drives from \(X^*\). This fact occurs at the initial stages of the search process because \(p\) is near to 0 and parametrised assortative mating tries to mate \(X^*\) with unlike individuals. On the opposite, when \(X^*\) and the mate are similar, offspring is sampled close to \(X^*\). It happens at last stages because \(p\) is near to 1 and parametrised assortative mating attempts coupling \(X^*\) with similar individuals.

This is a promising characteristic because Yao [64] theoretically proved that employing larger and dynamic neighbourhood operators may enhance efficiency of SA. In fact, in Liu [65], the impact of neighbourhood size on the performance of SA is analysed and another SA algorithm with dynamic neighbourhood sizes is proposed for flowshop scheduling problems. We shall mention as a note that, though the ideas in Yao [64] and in SASEA point out in the same direction, the study in Yao [64] does not explicitly cover crossover-based neighbourhood structures.

- **Neighbourhood exploration is guided by the mate to promote diversification and/or intensification:** at the first stage, crossover operator pushes \(Z\) from \(X^*\) toward the mate. Thus, mate guides the exploration of \(X^*\) neighbourhood. At this point, parametrised assortative mating becomes determinant: at initial stages of the search process, it intends to promote diversification by coupling \(X^*\) with dissimilar individuals; whereas at latter stages, it attempts promoting intensification by pairing \(X^*\) with similar individuals.

This characteristic has also been considered beneficial for the performance of SA in previous works. Fox [66] states that neighbourhood exploration might be guided by non-uniform probability distributions. Instead of this blind exploration process, he suggests that modified distributions might promote diversification and/or intensification. A particular example for continuous optimisation problems is found in [67]. The presented neighbourhood exploration favours diversification in the initial stages of the search process by sampling opposite neighbours.
5.2. Comparison with other SA neighbourhood structures

With the aim of analysing the significance of the neighbourhood structure of SASEA, we compare the results of the proposal with the ones of two other SA algorithms with different neighbourhood structures, classic SA and dynamic SA [65], on the experimental framework described in Section 4. Though SASEA was previously compared with classic SA in [29], the latter is included here for the sake of completeness.

Initial temperature was set in the following manner for all the search algorithms: firstly, two random solutions are generated. We set a desired probability $p_d$ of accepting the worst solution from the best one. Then, we compute the corresponding $T_0$ value, according to the applied acceptance criterion. This procedure, which has not been proposed before as far as we know, allows us to probabilistically get an appropriate value for the initial temperature according to the range of the fitness function and a desired initial probability for worse solutions, without consuming many evaluations.

Initial experiments comparing SASEA and classic SA showed that geometric cooling and any of the two acceptance mechanisms (metropolis or logistic) were the combinations that obtained the best results for both models, classic SA and SASEA [29]. Besides, best results were obtained with $p_d$ equal to 0.4, 100 iterations per cooling event, and 0.99 as the cooling factor. Therefore, all methods will apply geometric cooling, logistic acceptance mechanism, and the mentioned parameters values.

**Dynamic SA** applies the following formula [65] to compute the probability of flipping each bit of $X^*$:

$$1 + \left( \frac{N}{2} - 1 \right) \left( 1 - \frac{FES}{\text{Max,FES}} \right)^{0.4},$$

(5)

where $N$ is the number of bits of the problem, $FES$ is the number of evaluations consumed, and Max,FES is the maximum number of evaluations.

SASEA maintains 500 individuals in the population and $n_T$ is set to 15 for restricted tournament selection (parameter values taken from BLGA [39]). In addition, $p_{cr}$ for the crossover operator, has been set equal to $\frac{3}{N}$, and $n_{ass}$ to 10 for parameterised assortative mating.

**Tables 8–10,** in Appendix C, outline the results of the studied algorithms when tackling each test problem. **Table 2 summarises** the results of applying the Wilcoxon matched-pairs signed ranks test for $p$-value $= 0.05$, where the values of $R^+$ (associated to SASEA) and $R^-$ of the test are specified. If $R^+$ is smaller than $R^-$ and the critical value of the test, SASEA is statistically better than the corresponding algorithm; if $R^+$ is inferior to $R^-$ and the critical value, SASEA is statistically worse than its competitor; if neither $R^+$ nor $R^-$ is smaller than the critical value, Wilcoxon test does not find statistical differences ($p$-value $= 0.05$ and 27 problem instances make critical value to be 107). Last column indicates whether SASEA performs statistically better (+), worse (−), or without significant differences (−) than its competitor.

From **Table 2,** we clearly notice that the optimisation process performed by SASEA is statistically superior to the ones of the other two algorithms. It is also interesting to see that SASEA outperforms dynamic SA on almost every test problem because the associated $R^-$ value is close to 0.

We have carried out a statistical analysis for $p$-value equal to 0.05 to measure the performance differences between SASEA and

![Fig. 9. Convergence graphs of SA algorithms with different neighbourhood structures.](image)

the other algorithms on each problem separately. This statistical analysis is based on Mann–Whitney test [68]. Plus (+), minus (−), and approximate (−−) signs in Tables 8–10 indicate superior, inferior, or similar performance of SASEA, respectively, with regard to the corresponding algorithm on each problem. We notice that:

- SASEA obtains better results than the dynamic SA on almost all the problems considered. This fact is in accord with the results in Table 2.
- Classic SA and SASEA attain comparable results on many problems. However, SASEA shows a light superiority on several artificial problems (Royal road, Deceptive and Bipolar deceptive problems, and three PPeaks ones) and two Multiple knapsack ones. Classic SA achieves better results on just two problem instances (Overlapping deceptive problem and Maxcut(ising25-250,5555)).

Next, we study the behaviour of the three search algorithms along the whole run. In order to obtain a convergence graph summarising algorithms’ behaviours on all the problems, we have accomplished the following two steps:

1. Taking into consideration highest and lowest fitness values achieved by all the studied algorithms on each test problem, we have normalised every result, along the whole runs, into the interval [0, 1).
2. Subsequently, mean values, over the 27 problem instances, have been obtained for each algorithm, along the $10^5$ evaluations.

Fig. 9 shows the results for classic SA, dynamic SA, and SASEA. From Fig. 9, we see that the results of SASEA are superior to both SA algorithms along the whole run. Besides, we know that, at least at the end of the run, the differences are statistically significant (Table 2). On the other hand, we observe that dynamic SA lasts too much time before leading the search process toward promising solutions, which occurs when the number of flips on $X^*$ is low. This effect explains the poor results of dynamic SA in Tables 2 and 8–10.

The conclusion of this study is that the natural way SASEA implements the ideas commented in Section 5.1, dynamic neighbourhood sizes and guided neighbourhood exploration for diversification and/or intensification, becomes beneficial to outperform the results of other SA approaches.
5.3. Guiding neighbourhood and SASEA vs. HUX and CHC

In this section, we address the proper adjustment of parameter \( p_y \) of guiding neighbourhood exploration crossover operator for the proper operation of SASEA, and compare it with the operation of HUX in CHC. The motivation is that both crossover operators initially work according to the differences between the chosen parents, however, our initial experiments suggested to set \( p_y \) to 3/\( N \) (this is the probability of selecting genes from parent \( Y \)), whereas the implicit parameter in HUX considers a value of 0.5 (it exchanges exactly half of the differences between both parents) [69].

We have performed experiments according to the empirical framework described in Section 4, comparing SASEA and CHC with different parameter values for their crossover operators. In particular, we will compare the results of both algorithms with every combination of values for \( p_y \in \{ 3/N, 0.3, 0.5 \} \). Notice that we have implemented a HUX version that exchanges a number of differences between the parents that depends on the given parameter \( p_y \). The population of CHC consists of 50 individuals.

Table 3 presents every possible pairwise comparison between both algorithms by means of Wilcoxon statistical test with \( p \)-value = 0.05 (critical value is 107). Algorithms are denoted as SASEA(\( p_y \)) and CHC(\( p_y \)), respectively. \( R^+ \) and \( R^- \) values are presented for every comparison. If \( R^- \) is smaller than the critical value, then, Wilcoxon test finds statistical differences favouring SASEA.

No statistical differences favouring CHC were obtained. We observe that:

- SASEA(3/N) (which is the usual configuration) outperforms CHC regardless the parameter value considered.
- Though CHC(0.5) seems to outperform SASEA(0.5), Wilcoxon test does not find statistical differences.
- SASEA provides the best results when \( p_y = 3/N \) (because it outperforms all the CHC approaches), i.e., when \( X^k \) is perturbed locally, which is at the foundation of classic SA methods. However, CHC obtains its best results when \( p_y = 0.5 \) (it is more competitive with regard to the SASEA instances), which coincides with its original conception [43,69], i.e., when it performs a global search, which is as well at the core of evolutionary algorithms. Nevertheless, when both algorithms are tuned, Wilcoxon finds statistical differences favouring SASEA.

To sum up, we may conclude that \( p_y \) has to be tuned according to the demands of the algorithm: \( X^k \) must be modified locally for trajectory methods such as SASEA, whereas a global exploration has to be provided for methods such as CHC.

6. SASEA vs. other hybrid MHs based on SA and EAs

Nowadays, there exist several hybrid MH models combining ideas from the EA and SA fields. Usually, their aim is to improve the performance of SA when it is applied in a limited resources framework, i.e., a maximum allotted time, number of evaluations, etc. Several of these proposals are:

- **Parallel recombinative SA (PRSA) [22]**. It borrows the ideas from EAs to maintain several copies of SA running in parallel, with mutation as the neighbourhood operator, and crossover recombining independent solutions. PRSA may be seen as an EA, with random selection of parents, one-point-crossover and standard mutation operators, where offspring replace their parents according to the acceptance mechanism of SA.
- **Annealing-Genetic Algorithm (AGA) [24]**. It is a SA algorithm with the population-based state transition and with the genetic-operator-based quasi-equilibrium control. AGA can be understood as an EA where best solutions are optimised by SA processes. AGA selects parents based on their fitness values and applies two-point-crossover and mutation operators. Its inner SA procedure considers the metropolis acceptance mechanism. In addition, AGA includes a specific procedure to compute the initial temperature (\( T_0 \)), and cooling scheme is applied at every iteration.
- **SA, Genetic Algorithm, Chemotaxis algorithm, Integrated Algorithm (SAGACIA) [23]**. SAGACIA is based on proper integration of SA, Genetic Algorithm, and Chemotaxis Algorithm for solving complex optimisation problems. It combines three mechanisms: rough search, fine search and disturbance one. At the first stage, new solutions are sampled in the neighbourhoods of the members of the population; rough search usually makes large jumps in the search space; then, new solutions replace their parents according to the metropolis criterion. At the fine search stage, the best member of the population is optimised by a local search method with a budget of \( N \) evaluations (the number of bits of the problem). At the latter stage, the members of the population are disturbed with a low probability by a mutation operator.

Besides, we found other models that implement the idea of a population of SA processes, which makes them to be similar to the aforementioned hybrid MHs:

- **Sample-Sort SA (SSSA) [25]**. SSSA maintains an array of samplers operating at static temperatures. At each iteration, each sampler firstly considers to accept any of the states of its neighbouring samplers; biased acceptance mechanisms are applied at this phase; subsequently, each sampler performs a standard iteration of SA. Metropolis acceptance mechanism is applied. SSSA includes a specific procedure to compute the temperature of first and last samplers. Afterwards, the temperature values of the other samplers are set according to \( T_k = T_{last} \cdot \alpha^{m−k} \), where \( m \) is the number of samplers and \( \alpha \) was previously computed satisfying that \( T_{first} = T_{last} \cdot \alpha^m \).
- **Coupled SA (CSA) [26]**. It considers a population of samplers where the acceptance probability of any sampler making an uphill move depends on the states of the other samplers. In particular, when sampler \( i \) generates a new solution \( Y_i \) from the state \( X_i \), the probability of accepting such a move is:

\[
\exp \left( \frac{f(X_i) − max_{X \in \theta} f(X)}{T} \right)
\]

\[
γ = \sum_{X \in \theta} \exp \left( \frac{f(X) − max_{X \in \theta} f(X)}{T} \right)
\]

where \( \theta \) is the set of current states of the samplers. Notice that the acceptance probability does not depend on \( f(Y_i) \). Besides, CSA includes a procedure that controls the temperature of the system. The aim of this procedure is to maintain the variance of the acceptance probabilities of the samplers close to a desired variance value, which is computed as \( 0.99 \times (\lfloor samplers \rfloor − 1)/(\lfloor samplers \rfloor^2) \).
All the search algorithms described in this section emphasise the evolution of a set of solutions in parallel. Thus, the notion of evolving populations from EAs is the centre of attention and SA ideas just define some actions on its members. SASEA is a new alternative to this scheme for combining EAs and SA. Search process performed by SASEA focuses on just one solution (X) that wanders over the search space looking for the best configuration. SA ideas govern this process, whereas EA components (i.e. mating selection mechanism, crossover operator, and replacement strategy), which operate on a population of solutions, just modifies X’s neighbourhood exploration. Because of that fact, SASEA can be seen either as a SA approach based on a specialised EA or as a specialised EA that carries out the search process of SA.

Now, we are interested in determining whether SASEA really provides an improved alternative to the schemes for combining EAs and SA. In order to do this, we pit SASEA against aforementioned algorithms. The experimental framework is the one described in Section 4. We have followed the recommendations of the original publications for the parameters settings:

- **PRSA** applies a mutation probability equal to 1/N, which is high enough for the mutation operator to be regarded as the neighbourhood operator. Population size is set to 60, which is a standard value for EAs. On the other hand, the remainder design decisions were made as for SASEA: PRSA applies logistic acceptance criterion, geometric cooling with α set to 0.99, 100 iteration per cooling event, and p_d = 0.4.
- **AGA** uses a mutation probability equal to 1/(10 - N), which is as low as it occurs in EAs. Population size is 60. In addition, it applies geometric cooling with α equal to 0.99 as SASEA.
- **SAGACIA** uses a population with 60 individuals. Mutation probability for the rough search stage is set to 10/N, with the aim of producing large jumps. Mutation probability for the disturbance stage is 0.025. T_c is computed as for SASEA with p_d set to 0.4.
- **SSSA** maintains 100 samplers. Neighbourhood size is 1, i.e., each sampler may accept any of the two states of its consecutive samplers (just one for the extreme samplers).
- **CSA** has 5 samplers. Its parameter α, for the variance control procedure, is set to 0.05. In addition, its acceptance probability formulae has been adapted for maximisation problems as:

  \[
  \gamma = \sum_{x \in \Theta} \exp \left( \frac{\min_{x \in \Theta} (f(X)) - f(x)}{T} \right),
  \]

  \[
  \gamma = \sum_{x \in \Theta} \exp \left( \frac{\min_{x \in \Theta} (f(X)) - f(x)}{T} \right),
  \]  

  \[
  \gamma = \sum_{x \in \Theta} \exp \left( \frac{\min_{x \in \Theta} (f(X)) - f(x)}{T} \right),
  \]

Tables 8–10, in Appendix C, contain the results of the search algorithms when tackling each test problem. Table 4 summarises the results of applying the Wilcoxon matched-pairs signed ranks test for p-value = 0.05. The superiority of SASEA is clearly seen on the results presented in Table 4. Hence, the alternative scheme for combining ideas from SA and EAs introduced by SASEA becomes a promising research line to improve the performance of this kind of hybrid MHs. In particular, these results show that a single SA process consuming the allowed number of evaluations may obtain better results than a set of SA processes that compete for the computation resources.

Regarding the results of the algorithms on the problems separately (see Tables 8–10), we notice that SASEA generally outperforms its competitors, with the following exceptions:

- **AGA** attains similar results to those of SASEA on problems NkLand(48,4), BQP(bqp50-1), and Multiple knapsack problems (pb4). It is interesting to see that SASEA obtains the best results for those three problems and many algorithms get similar results (7, 6, and 10 out of 12 algorithms, respectively). Therefore, these problems seem to be easy to optimise. On the other hand, AGA outperforms SASEA on just one problem, HIFF(2,5,true).
- **SAGACIA** achieves the good results of SASEA on the mentioned easy problems and two other ones, M-Sat(100, 1200, 3) and Maxcut(pm1s80.6). Moreover, SAGACIA outperforms SASEA on five problems: Overlapping deceptive problem, M-Sat(100, 2400, 3), and Maxcut problems w09-100.2, g05-100.5, and pw05-100.6. These results are in accordance with the ones in Table 4 and SAGACIA appears as the most competitive algorithm with regard to SASEA among these hybrid MHs. In particular, SAGACIA obtains better results than SASEA on three out of five Maxcut problems, and it is surpassed on just one of them.

Fig. 10 shows convergence graphs for the studied algorithms, obtained by the same procedure described in Section 5.2. We see that most of the algorithms early get trapped in local optima or suffer premature convergence to poor solutions. Then, they have difficulties to improve further. However, the search process of SASEA is quietly driven toward promising solutions, overcoming local optima and reaching better final results than the other methods.

It is also interesting to notice that the graph of AGA surpasses the one of SAGACIA, though its results were poorer according to the statistical tests applied (Tables 4 and 8–10). The reason is that there are several problems for which AGA gets better results than SAGACIA, but not enough to outperform SASEA. On the contrary, SAGACIA really outperforms SASEA on several problems, but obtains results poorer than AGA on another ones (compare the results of AGA and SAGACIA on Royal road, HIFF, problems, and Maxcut(ising2.5-250.5)), for instance.

Table 4
SASEA vs. other hybrid MH (Wilcoxon’s test with p-value = 0.05 and critical value = 107).

<table>
<thead>
<tr>
<th>Algorithms</th>
<th>R⁺</th>
<th>R⁻</th>
<th>Sig. differences?</th>
</tr>
</thead>
<tbody>
<tr>
<td>SASEA vs. PRSA</td>
<td>378</td>
<td>0</td>
<td>+</td>
</tr>
<tr>
<td>SASEA vs. AGA</td>
<td>354.5</td>
<td>23.5</td>
<td>+</td>
</tr>
<tr>
<td>SASEA vs. SAGACIA</td>
<td>312.5</td>
<td>65.5</td>
<td>+</td>
</tr>
<tr>
<td>SASEA vs. SSSA</td>
<td>377.5</td>
<td>0.5</td>
<td>+</td>
</tr>
<tr>
<td>SASEA vs. CSA</td>
<td>378</td>
<td>0</td>
<td>+</td>
</tr>
</tbody>
</table>
7. SASEA vs. state-of-the-art algorithms for binary combinatorial problems

In this section, we intend to assess the performance of SASEA with regard to relevant optimisers for binary-coded problems found in the literature:

- **Cross-generational elitist selection, Heterogeneous recombination, and Catastrophic mutation (CHC)** [43]. Though this algorithm was already considered in Section 5.3, it is included here for the sake of completeness. CHC involves the combination of a selection strategy with a very high selective pressure and several components inducing diversity. CHC was tested against different Genetic Algorithms, giving better results especially on hard problems [69]. So, it has arisen as a reference point in the literature. Its population consists of 50 individuals.

- **Variable Dissortative Mating Genetic Algorithm (VDMGA)** [41]. It is a recent steady-state EA, similar to CHC, in which the number of new chromosomes entering the population in each generation is controlled on-line by a threshold value, genetic diversity, and population’s state of convergence. The results in [41] displayed the superior performance of VDMGA when compared to other Genetic Algorithms in which CHC was included. Its population size parameter is set to 100. In addition, the mutation probability is set to 0.006.

- **Sawtooth Genetic Algorithm (Saw-GA)** [70]. It is a Genetic Algorithm that uses a variable population size and periodic partial reinitialisation of the population in the form of a saw-tooth function. For a wide range of problems, the performance of Saw-GA was statistically superior to a standard and a micro Genetic Algorithm. The average population size is set to 80; crossover and mutation probabilities are set to 0.85 and 0.05, respectively; period and amplitude parameters are adjusted to 40 and 75, respectively.

- **Context-Independent Scatter Search (CISS)** [71]. CISS is a proposal explicitly designed to tackle general binary-coded problems. Its performance was compared with the one of several general-purpose commercial optimisation tools, obtaining promising results. Population and reference set sizes have been set to 300 and 6, respectively. The other parameters have been set as in [71].

- **Versatile Quantum-inspired EA (vQEA)** [72]. vQEA is a recent EA approach based on quantum computing principles. It considers the quantum bit (Qbit) as the smallest information unit, which is defined by the probability at which the corresponding state (0 or 1) is likely to appear when it is collapsed, i.e., read or measured. vQEA considers a population of quantum individuals (a quantum individual is composed of a Qbit string, a realization, and an attractor) that evolve through quantum gate operations. The population of vQEA is divided into groups each containing d individuals. Attractors are periodically synchronized between individuals in the same group and between different groups. vQEA considers a population of 10 individuals distributed into an unique group. Attractors synchronization takes place every generation and Δθ, associated to quantum gate, is \( \pi/100 \).

Tables 8–10, in Appendix C, show the results of the search algorithms when tackling each test problem. Table 5 summarises the results of applying the Wilcoxon matched-pairs signed rank test for p-value = 0.05. We notice that SASEA obtains results statistically better than the ones of the other optimisers. In addition, differences between \( R^+ \) and \( R^- \) values are relevant. Therefore, SASEA arises as a promising tool to tackle binary-coded problems.

Taking a deeper look into Tables 8–10, we see that:

- Saw-GA is outperformed by SASEA on almost all the functions. The statistical test does not find significant differences between their results on just 5 problems, which are Trap, Deceptive, NkLand(48, 4), and two Multiple knapsack problems, weish03 and pb4. An interesting fact is that the number of variables of those problems is significantly low with regard to the ones of the other problems.

- SASEA outperforms CHC on 17 out of the 27 problem instances considered. In particular, SASEA generally shows superiority on the M-sat, Maxcut, BQP, and Knapsack real-world problems. On the other hand, CHC obtains statistically better results than SASEA on two problems, Bipolar deceptive and PPeaks(10,100), and it proves to be competitive with regard to SASEA on several another ones: Trap, NkLand problems, HIFF(2, 5, true), PPeaks(100,100), BQP(bqp50-1), Maxcut(ising2.5-250.5555), and Multiple knapsack problem pb4. According to these results, we did not find a specific class of problems where CHC appeared superior to SASEA.

- VDMGA attains similar results to those of SASEA on many problems, however, it only achieves better results on just one problem, Royal road. On the other hand, SASEA is superior to VDMGA on M-sat, Maxcut, and BQP real-world problems.

- We see that SASEA achieves better results than CISS on several artificial problems such as HIFF, Royal road, Trap, Deceptive and Bipolar deceptive, and on real-world ones such as BQP and Maxcut, too. On the other hand, CISS shows superiority on the artificial NkLand problems, as well as on the Overlapping deceptive one. In our opinion, its particular mechanisms to protect and enhance diversification are the reasons for its good results on those commonly said complex problems. Both algorithms obtain similar results on all the PPeaks and Multiple knapsack problems.

- Finally, SASEA outperforms vQEA almost on every function. vQEA just obtains similar results to those of SASEA on three PPeaks problems.

Fig. 11 shows convergence graphs for the studied optimisers. They were obtained by the same procedure described in Section 5.2. The behaviour of the search algorithms are very similar to those of the hybrid MHs in Section 6. Most of the methods seem to
get trapped in local optima early or experience slow convergence toward good solutions. However, though SASEA reaches solutions of equal quality later, its ability to protect diversification at the initial stages of the process and gradually increase intensification, by means of the proposed mating mechanism, crossover operator, and replacement strategy, lets SASEA to overcome local optima and reach better results.

8. Conclusion

In this work, we proposed SASEA, an innovative hybrid MH based on a specialised EA that executes a search process equivalent to the one of SA. We have performed several experimental studies concluding that:

- The neighbourhood structure associated to the components of SASEA, which dynamically adapts its size and can be used to promote intensification and/or diversification, becomes beneficial when compared to the classic neighbourhood structure (one-flip) of SA and another one presented in the literature that dynamically adapts its size.
- The alternative scheme for combining ideas from SA and EAs introduced by SASEA may outperform other hybrid MHs based on SA and EAs.
- SASEA provides a significantly better performance than other state-of-the-art algorithms for binary-coded problems, on the test suite considered.

The research line focused in this paper is indeed worth of further studies. We are currently extending our investigation to build evolutionary models incorporating SASEA to perform a search process on a set of points in parallel, and to take benefits from parallel hardware. In addition, we intend to extend our investigation to different test-suits, other coding schemes and other real-world problems, such as training set selection in data mining [73].

Acknowledgements

This work was supported by Research Projects TIN2008-05854 and P08-TIC-4173.

Appendix A. Test suite

The test suite that we have used for the experiments consists of 27 binary-coded test problem instances from 12 different problem classes, 8 classes (13 instances) from the artificial intelligence field and 4 (14 instances) from real-world applications. They are described in the following sections.

A.1. Royal road

To construct a royal road problem [74], we select a random optimum string and break it up into a number of small building blocks. We then assign values to each low-order schema and use those values to compute the fitness of a bit string X in terms of the schemes of which it is an instance. The value of each low-order schema is equal to its size. We have used a royal road problem with length 400 where low-order schemes consist of groups of 8 consecutive bits.

A.2. Trap problem

Trap problem [36] consists of misleading subfunctions of different lengths. Specifically, the fitness function f(x) is constructed by adding subfunctions of length 1 (F1, F2), and 3 (F3). Each subfunction has two optima: the optimal fitness value is obtained for an all-ones string, while the all-zeroes string represents a local optimum. The fitness of all other string in the subfunction is determined by the number of zeros: the more zeros, the higher the fitness value. This causes a large basin of attraction toward the local optimum. The fitness values for the subfunctions are specified in Table 6 where the columns indicate the number of ones in the subfunctions F1, F2, and F3. The fitness function f(x) is composed of 4 F3 subfunctions, 6 F2 subfunctions, and 12 F1 subfunctions. The overall length of the problem is thus 36. There are 210 optima of which only one is the global optimum: the string with all ones having a fitness value of 220.

\[ f(x) = \sum_{i=0}^{3} F_3(u_{3i+3,2i+2}) + \sum_{i=0}^{5} F_2(u_{2i+12,2i+13}) + \sum_{i=0}^{11} F_1(u_{24+i}) \]  

(A.1)

A.3. Deceptive problems

In deceptive problems [75], there are certain schemata that guide the search toward some solution that is not globally competitive. The schemata that have the global optimum do not bear significance and so they may not proliferate during the genetic process. The used deceptive problem consists of the concatenation of k subproblems of length 3. The fitness for each 3-bit section of the string is given in Table 7. The overall fitness is the sum of the fitness of these deceptive subproblems. We have used a deceptive problem with 13 subproblems.

A.4. Bipolar deceptive problem

A deceptive function of order 3 is defined as

\[ f_{\text{deceptive}}^3 = \begin{cases} 
0.9 & \text{if } u = 0 \\
0.8 & \text{if } u = 1 \\
0 & \text{if } u = 2 \\
1 & \text{otherwise}
\end{cases} \]  

(A.2)

where \( u \) is the sum of three input bits.

A bipolar deceptive function of order 6 [76] is defined with the use of deceptive subproblems of length 3 as follows:

\[ f_{\text{bipolar}}^6(X) = f_{\text{deceptive}}^3([3 - u]), \]  

(A.3)

where \( X \) is a vector of 6 binary variables, and \( u \) is the sum of the input bits. We have included a bipolar problem with 66 \( f_{\text{bipolar}}^6 \) subproblems. The overall fitness is the average of the fitness of these subproblems.

A.5. Overlap deceptive problem

A deceptive function composed of deceptive functions of order 3 (previous section) that are overlapping in one bit in a chain-like
structure will be referred to as 3-deceptive with overlapping [76]. It is defined as follows:

$$f_{\text{dec-overlap}}(X) = \sum_{i=0}^{(n-3)/2} f_{\text{deceptive}}(S_i).$$  \hfill (A.4)

where \(X = (X_0, \ldots, X_{n-1})\) is a vector of bits, and \(S_i\) is the concatenation of bits \(X_{2i}, X_{2i+1}\), and \(X_{2i+2}\). The overlap deceptive problem included in the test suite consists of 199 \(f_{\text{dec-overlap}}\) subfunctions, making the length of the problem to be equal to 399. Overall fitness is the average of the fitness of the subfunctions.

### A.6. NK-Landscapes

In the NK model [77], \(N\) represents the number of genes in a haploid chromosome and \(K\) represents the number of linkages each gene has to other genes in the same chromosome. To compute the fitness of the entire chromosome, the fitness contribution from each locus is averaged as follows:

$$f(s) = \frac{1}{N} \sum_{i=1}^{N} f(\text{locus}_i)$$  \hfill (A.5)

where the fitness contribution of each locus, \(f(\text{locus}_i)\), is determined by using the (binary) value of gene \(i\) together with values of the \(K\) interacting genes as an index into a table \(T_i\) of size \(2^{K+1}\) of randomly generated numbers uniformly distributed over the interval \([0, 1]\). For a given gene \(i\), the set of \(K\) linked genes may be randomly selected or consist of the immediately adjacent genes.

We have used two sets of instances of the NK-Landscape problem: one with \(N = 48\) and \(K = 4\), and another with \(N = 48\) and \(K = 12\). They are denoted as NKLand\((N, K)\). They have been obtained from [78]. Each run \(i\) of every search algorithm, uses a different seed (seed\(_i\)) for generating the NKLand\((N, K)\) instance, i.e. the \(i\)th execution of every method has used the same seed\(_i\), whereas the \(j\)th execution has used seed\(_j\).

### A.7. Hierarchical if-and-only-if problems

Hierarchical if-and-only-if problems (HIFF) [79] are defined as follows: A binary string \(X = (x_1, \ldots, x_n)\), where \(n = kp\), represents a hierarchical block structure, where \(k\) is the number of sub-blocks in a block, and \(p\) is the number of levels in the hierarchy. Blocks do not need to consist of consecutive sub-blocks, i.e. they can be shuffled. Each block at the bottom level of this hierarchy, consisting of \(k\) bits each, will be converted into a single symbol by a transform function, \(t\). This transform function defines the meaning of each block. This creates a new string (with length \(k^{l-1}\)) that is the decoding of the block structure to the first level. This process is repeated for each level in the hierarchy to give the single symbol that is the meaning of the whole structure. Thus, the recursive transform function, \(T\), transforms any block structure to its meaning, a single symbol:

$$T(X) = \begin{cases} x_i & \text{if } |X| = 1, \\ t(T(X^1), \ldots, T(X^k)) & \text{otherwise} \end{cases}$$  \hfill (A.6)

where \(t\) is a base function that defines the resultant symbol from a block of \(k\) symbols and \(X^i\) is the \(i\)th sub-block of \(X\), i.e. \(\{x_{i-1}d^i+1, \ldots, x_{id}\}\) where \(d = |B|/k\).

Now we may use \(T(X)\) to construct \(f(X)\), the fitness of a block structure. Specifically, the fitness of a block structure will be the fitness contribution of its transform (scaled for its size) plus the sum of the fitness of its sub-blocks. Hence the recursive function, \(f\), defined using the base function \(F\):

$$f(X) = \begin{cases} F(X) & \text{if } |X| = 1, \\ |X|F(T(B)) + \sum_{i=1}^{k} f(X^i) & \text{otherwise} \end{cases}$$  \hfill (A.7)

where \(F\) is a base function giving the fitness of a single bit.

We now give the base functions, \(F\) and \(t\), that provide an interesting fitness landscape. First we define \(t(A, B)\) by arbitrarily assigning 0 and 1 to the two solutions of IFF (i.e. \(t(0, 0) = 0, t(1, 1) = 1\)) and null for any other combination of 0, 1, and null. \(F(A)\) naturally defines the two non-null transform values as desirable and null as undesirable.

We have used two instances of this problem. The first one has 5 levels and 2 shuffled subblocks per block, and the second one, 4 levels and 3 consecutive subblocks per block.

### A.8. P-Peak problems

P-Peak problem generator [80] creates instances with a certain number of peaks (the degree of multi-modality). For a problem with \(P\) peaks, \(P\) bit strings of length \(L\) are randomly generated. Each of these strings is a peak (a local optima) in the landscape. Different heights can be assigned to different peaks based on various schemes (equal height, linear, logarithm-based, and so on). To evaluate an arbitrary solution \(S\), first locate the nearest peak in Hamming space, call it Peak\(_0(S)\). Then, the fitness of \(S\) is the number of bits the string has in common with Peak\(_0(S)\), divided by \(L\), and scaled by the height of the nearest peak. In case there is a tie when finding the nearest peak, the highest peak is chosen.

We have used different groups of P-Peak instances denoted as PPeaks\((P, L)\). Each run \(i\) of every search algorithm, uses a different seed (seed\(_i\)) for generating the PPeaks\((P, L)\) instance. Linear scheme have been used for assigning heights to peaks in [0.6, 1].

### A.9. Max-Sat problem

The satisfiability problem in propositional logic (SAT) [81] is the task to decide whether a given propositional formula has a model. More formally, given a set of \(m\) clauses \(\{C_1, \ldots, C_m\}\) involving \(n\) boolean variables \(X_1, \ldots, X_n\) the SAT problem is to decide whether an assignment of values to variables exists such that all clauses are simultaneously satisfied.

Max-Sat is the optimisation variant of SAT and can be seen as a generalisation of the SAT problem: Given a propositional formula in conjunctive normal form (CNF), the Max-Sat problem is to find a variable assignment that maximises the number of satisfied clauses. It returns the percentage of satisfied clauses.

We have used two sets of instances of the Max-Sat problem with 100 variables, 3 variables by clause, and 1200 and 2400 clauses, respectively. They have been obtained from [78]. They are denoted as M-Sat\((n, m, l)\), where \(l\) indicates the number of variables involved in each clause (3). Each run \(i\) of every search algorithm, uses a specific seed (seed\(_i\)) for generating the M-Sat\((n, m, l)\) instance, i.e. \(i\)th execution of every method uses the same seed\(_i\), whereas \(j\)th execution uses seed\(_j\).

### A.10. Unconstrained Binary Quadratic Programming problem

The objective of the Unconstrained Binary Quadratic Programming (BQP) [82] is to find, given a symmetric rational \(n \times n\) matrix
design with two samples. It is a pairwise test that aims to detect significant differences between the behaviour of two algorithms.

The null hypothesis for Wilcoxon’s test is \( H_0 : \theta_2 = 0 \); in the underlying populations represented by the two samples of results, the average of the difference scores equals zero. The alternative hypothesis is \( H_1 : \theta_2 \neq 0 \), but also can be used \( H_1 : \theta_2 > 0 \) or \( H_1 : \theta_2 < 0 \) as directional hypothesis.

In the following, we describe the tests computations. Let \( d_i \) be the difference between the performance scores of the two search algorithms on \( i \)th out of \( N \) functions. The differences are ranked according to their absolute values (we have previously normalised the results of every algorithm on each test problem into the interval \([0, 1]\), taking into consideration highest and lowest fitness values achieved on each test problem); average ranks are assigned in case of ties. Let \( R^+ \) be the sum of ranks for the functions on which the second algorithm outperformed the first, and \( R^- \) the sum of ranks for the opposite. Ranks of \( d_i = 0 \) are split evenly among the sums; if there is an odd number of them, one is ignored:

\[
R^+ = \sum_{d_i > 0} \text{rank}(d_i) + \frac{1}{2} \sum_{d_i = 0} \text{rank}(d_i) \quad \text{and} \quad R^- = \sum_{d_i < 0} \text{rank}(d_i) + \frac{1}{2} \sum_{d_i = 0} \text{rank}(d_i) \quad \text{(B.1)}
\]

Let \( T \) be the smallest of the sums, \( T = \min(R^+, R^-) \). If \( T \) is less than or equal to the value of the distribution of Wilcoxon for \( N \) degrees of freedom (Table B.12 in [84]); the null hypothesis of equality of means is rejected.

The obtaining of the \( p \)-value associated to a comparison is performed by means of the normal approximation for the Wilcoxon \( T \) statistic (Section VI, Test 18 in [85]). Furthermore, the computation of the \( p \)-value for this test is usually included in well-known statistical software packages (SPSS, SAS, R, etc.).

**Appendix C. Results**

Tables 8–10 show the average of fitness values obtained by the studied search algorithms. Best results for each problem are boldfaced. Moreover, we have carried out a statistical analysis with \( p \)-value equal to 0.05 to measure the performance differences between SASEA and the other algorithms on each problem separately. This statistical analysis is based on Mann–Whitney test [68]. A plus sign (+) indicates that the performance of SASEA is superior to the corresponding algorithm on that problem. A minus sign (−) means that the performance of SASEA is inferior to the corresponding algorithm on that problem. An approximate sign (≈) is written when the tests did not find significant differences between the performances of SASEA and the corresponding algorithm on the problem.
### Table 8
Results of the search algorithms on test problems 1–9.

<table>
<thead>
<tr>
<th>Alg</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>PRSA</td>
<td>0.086</td>
<td>0.571</td>
<td>0.272</td>
<td>0.374</td>
<td>0.272</td>
<td>0.151</td>
<td>0.117</td>
<td>0.272</td>
<td>0.272</td>
</tr>
<tr>
<td>AGA</td>
<td>0.086</td>
<td>0.571</td>
<td>0.272</td>
<td>0.374</td>
<td>0.272</td>
<td>0.151</td>
<td>0.117</td>
<td>0.272</td>
<td>0.272</td>
</tr>
<tr>
<td>SA</td>
<td>0.086</td>
<td>0.571</td>
<td>0.272</td>
<td>0.374</td>
<td>0.272</td>
<td>0.151</td>
<td>0.117</td>
<td>0.272</td>
<td>0.272</td>
</tr>
<tr>
<td>SSA</td>
<td>0.086</td>
<td>0.571</td>
<td>0.272</td>
<td>0.374</td>
<td>0.272</td>
<td>0.151</td>
<td>0.117</td>
<td>0.272</td>
<td>0.272</td>
</tr>
<tr>
<td>SSA</td>
<td>0.086</td>
<td>0.571</td>
<td>0.272</td>
<td>0.374</td>
<td>0.272</td>
<td>0.151</td>
<td>0.117</td>
<td>0.272</td>
<td>0.272</td>
</tr>
<tr>
<td>SSA</td>
<td>0.086</td>
<td>0.571</td>
<td>0.272</td>
<td>0.374</td>
<td>0.272</td>
<td>0.151</td>
<td>0.117</td>
<td>0.272</td>
<td>0.272</td>
</tr>
<tr>
<td>SSA</td>
<td>0.086</td>
<td>0.571</td>
<td>0.272</td>
<td>0.374</td>
<td>0.272</td>
<td>0.151</td>
<td>0.117</td>
<td>0.272</td>
<td>0.272</td>
</tr>
<tr>
<td>SSA</td>
<td>0.086</td>
<td>0.571</td>
<td>0.272</td>
<td>0.374</td>
<td>0.272</td>
<td>0.151</td>
<td>0.117</td>
<td>0.272</td>
<td>0.272</td>
</tr>
<tr>
<td>SSA</td>
<td>0.086</td>
<td>0.571</td>
<td>0.272</td>
<td>0.374</td>
<td>0.272</td>
<td>0.151</td>
<td>0.117</td>
<td>0.272</td>
<td>0.272</td>
</tr>
</tbody>
</table>

### Table 9
Results of the search algorithms on test problems 10–18.

<table>
<thead>
<tr>
<th>Alg</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
<th>17</th>
<th>18</th>
</tr>
</thead>
<tbody>
<tr>
<td>PRSA</td>
<td>0.086</td>
<td>0.571</td>
<td>0.272</td>
<td>0.374</td>
<td>0.272</td>
<td>0.151</td>
<td>0.117</td>
<td>0.272</td>
<td>0.272</td>
</tr>
<tr>
<td>AGA</td>
<td>0.086</td>
<td>0.571</td>
<td>0.272</td>
<td>0.374</td>
<td>0.272</td>
<td>0.151</td>
<td>0.117</td>
<td>0.272</td>
<td>0.272</td>
</tr>
<tr>
<td>SA</td>
<td>0.086</td>
<td>0.571</td>
<td>0.272</td>
<td>0.374</td>
<td>0.272</td>
<td>0.151</td>
<td>0.117</td>
<td>0.272</td>
<td>0.272</td>
</tr>
<tr>
<td>SSA</td>
<td>0.086</td>
<td>0.571</td>
<td>0.272</td>
<td>0.374</td>
<td>0.272</td>
<td>0.151</td>
<td>0.117</td>
<td>0.272</td>
<td>0.272</td>
</tr>
<tr>
<td>SSA</td>
<td>0.086</td>
<td>0.571</td>
<td>0.272</td>
<td>0.374</td>
<td>0.272</td>
<td>0.151</td>
<td>0.117</td>
<td>0.272</td>
<td>0.272</td>
</tr>
<tr>
<td>SSA</td>
<td>0.086</td>
<td>0.571</td>
<td>0.272</td>
<td>0.374</td>
<td>0.272</td>
<td>0.151</td>
<td>0.117</td>
<td>0.272</td>
<td>0.272</td>
</tr>
<tr>
<td>SSA</td>
<td>0.086</td>
<td>0.571</td>
<td>0.272</td>
<td>0.374</td>
<td>0.272</td>
<td>0.151</td>
<td>0.117</td>
<td>0.272</td>
<td>0.272</td>
</tr>
<tr>
<td>SSA</td>
<td>0.086</td>
<td>0.571</td>
<td>0.272</td>
<td>0.374</td>
<td>0.272</td>
<td>0.151</td>
<td>0.117</td>
<td>0.272</td>
<td>0.272</td>
</tr>
<tr>
<td>SSA</td>
<td>0.086</td>
<td>0.571</td>
<td>0.272</td>
<td>0.374</td>
<td>0.272</td>
<td>0.151</td>
<td>0.117</td>
<td>0.272</td>
<td>0.272</td>
</tr>
</tbody>
</table>

### Table 10
Results of the search algorithms on test problems 19–27.

<table>
<thead>
<tr>
<th>Alg</th>
<th>19</th>
<th>20</th>
<th>21</th>
<th>22</th>
<th>23</th>
<th>24</th>
<th>25</th>
<th>26</th>
<th>27</th>
</tr>
</thead>
<tbody>
<tr>
<td>PRSA</td>
<td>0.086</td>
<td>0.571</td>
<td>0.272</td>
<td>0.374</td>
<td>0.272</td>
<td>0.151</td>
<td>0.117</td>
<td>0.272</td>
<td>0.272</td>
</tr>
<tr>
<td>AGA</td>
<td>0.086</td>
<td>0.571</td>
<td>0.272</td>
<td>0.374</td>
<td>0.272</td>
<td>0.151</td>
<td>0.117</td>
<td>0.272</td>
<td>0.272</td>
</tr>
<tr>
<td>SA</td>
<td>0.086</td>
<td>0.571</td>
<td>0.272</td>
<td>0.374</td>
<td>0.272</td>
<td>0.151</td>
<td>0.117</td>
<td>0.272</td>
<td>0.272</td>
</tr>
<tr>
<td>SSA</td>
<td>0.086</td>
<td>0.571</td>
<td>0.272</td>
<td>0.374</td>
<td>0.272</td>
<td>0.151</td>
<td>0.117</td>
<td>0.272</td>
<td>0.272</td>
</tr>
<tr>
<td>SSA</td>
<td>0.086</td>
<td>0.571</td>
<td>0.272</td>
<td>0.374</td>
<td>0.272</td>
<td>0.151</td>
<td>0.117</td>
<td>0.272</td>
<td>0.272</td>
</tr>
<tr>
<td>SSA</td>
<td>0.086</td>
<td>0.571</td>
<td>0.272</td>
<td>0.374</td>
<td>0.272</td>
<td>0.151</td>
<td>0.117</td>
<td>0.272</td>
<td>0.272</td>
</tr>
<tr>
<td>SSA</td>
<td>0.086</td>
<td>0.571</td>
<td>0.272</td>
<td>0.374</td>
<td>0.272</td>
<td>0.151</td>
<td>0.117</td>
<td>0.272</td>
<td>0.272</td>
</tr>
<tr>
<td>SSA</td>
<td>0.086</td>
<td>0.571</td>
<td>0.272</td>
<td>0.374</td>
<td>0.272</td>
<td>0.151</td>
<td>0.117</td>
<td>0.272</td>
<td>0.272</td>
</tr>
<tr>
<td>SSA</td>
<td>0.086</td>
<td>0.571</td>
<td>0.272</td>
<td>0.374</td>
<td>0.272</td>
<td>0.151</td>
<td>0.117</td>
<td>0.272</td>
<td>0.272</td>
</tr>
</tbody>
</table>

### References


