Bayesian Classifiers in Optimization: An EDA-like Approach

Teresa Miquélez[1], Endika Bengoetxea[1] and Pedro Larrañaga[2]

[1] Department of Computer Architecture and Technology, University of the Basque Country, P.O. Box 649, 20080 San Sebastian, Spain
{teresa,endika}@si.ehu.es

[2] Department of Computer Science and Artificial Intelligence, University of the Basque Country, P.O. Box 649, 20080 San Sebastian, Spain
ccplamup@si.ehu.es

Summary. This chapter introduces a new Evolutionary Computation method which applies Bayesian classifiers in the construction of a probabilistic graphical model that will be used to evolve a population of individuals. On the other hand, the *satisfiability* problem (SAT) is a central problem in the theory of computation as a representative example of NP-complete problems. We have verified the performance of this new method for the SAT problem. We compare three different solution representations suggested in the literature. Finally, we apply local search methods for this problem.

1 Introduction

This chapter introduces Evolutionary Bayesian Classifier-based Optimization Algorithms (EBCOAs) as a new approach in Evolutionary Computation. The originality of these algorithms comes from the fact that they evolve a generation of individuals by constructing Bayesian classifier models that take into account deeper differences rather than simply a subset of the better individuals of the previous population. The main difference between this approach and Estimation of Distribution Algorithms (EDAs) is the fact that the probabilistic graphical model in discrete EDAs is a Bayesian network, while in EBCOAs we construct a Bayesian classifier that includes an extra C node that represents the different classes to which each individual of a population is classified. EBCOAs take into account the differences between the individuals in the population that make them be more or less fit regarding their fitness value, and apply this knowledge to create a new population by enhancing the characteristics of the fitter ones and tries to avoid such of the less fit ones. In order to better understand the motivation for EBCOAs, this chapter analyzes the issues that allows Estimation of Distribution Algorithms (EDAs) to converge to the best solution of a problem, as well as several new methods for improving the way in which this convergence is done. The aim of this idea is to avoid a too fast convergence that could lead to fall in local optima.

In order to analyze the potential of this new method, we tried its performance with a typical NP-hard optimization algorithm such as the SAT problem. This optimization problem is regarded as a very interesting one in the field of computer science due to the big number of problems that can be formulated and solved using SAT instances.

The rest of the chapter is structured in the following way. Section 2 is devoted to the introduction to EBCOAs and the different Bayesian classifiers that can be applied. The SAT problem is defined in Sect. 3. Section 4 shows the experimental results obtained with this new method, and the conclusions of the chapter as well as ideas for future work can be found in Sect. 5.

2 The Evolutionary Bayesian Classifier-based Optimization Algorithm Approach

This section describes the EBCOAs [22]. Similarly as EDAs, EBCOAs combine both probabilistic reasoning and evolutionary computing. The main characteristic of EBCOAs that distinguish them from EDAs is that the learning of the probabilistic graphical model is based on using Bayesian classifiers.

2.1 Motivation

In many Evolutionary Computation techniques such as Genetic Algorithms (GAs) and EDAs only the best individuals of the generation are taken into account to proceed to apply crossover and mutation techniques–in GAs – or to learn the probabilistic graphical model – in EDAs. In these approaches, the aim is to take into account the characteristics of the N fittest individuals of the population. However, in most of the cases the fitness value differences among the individuals are not taken into account but for the purpose of deciding whether the phenotype of an individual is relevant for generating the next generation. Therefore, in the case of most EDAs, the best and worst individuals within the selected population of the $l\mathrm{th}$ generation are considered to have the same relevance for the learning of the probabilistic graphical model.

However, in many optimization problems the fitness differences between the selected individuals are also important in order to ensure an adequate convergence of the search process. This is essential in the case of EDAs in order to ensure convergence. The literature shows different possibilities for taking into account the fitness of each of the selected individuals in the learning step of EDAs:

- **Making fitter individuals to influence the learning of the probabilistic graphical model regarding their fitness value:** this approach assigns a weight to each individual to have more influence in the learning step in EDAs regarding the respective fitness value. An example of this idea was proposed in BSC [26].
- **Applying a proportional selection method:** An example of this approach is the use of a Boltzman distribution based selection [24].

- **Considering the fitness value as an additional node in the probabilistic graphical model:** the fact of including such a new variable together with variables $X_1, \ldots, X_n$ makes a direct influence on the learning and sampling steps of EDAs. Unfortunately, this fitness value variable is typically continuous and it is therefore difficult to apply learning algorithms that are able to handle at the same time discrete and continuous variables. In addition, these learning procedures are computationally more complex and require considerable CPU time.
- **Transform the learning of the probabilistic graphical model into a supervised classification problem:** this approach is the one proposed in EBCOAs, in which all the individuals of a population are classified in different classes, and then these are used to build Bayesian classifiers that have the form of a Bayesian network. These Bayesian networks have the characteristic of including the class-variable in the probabilistic graphical model as the parent of the rest of the variables $X_1, \ldots, X_n$. The main idea of this approach is to guide the search taking into account both the genotypes of the fittest and the less fit individuals.

EBCOAs are not the only approach in the literature that apply classification techniques in optimization. The most relevant statistical approach that follows a similar idea is the Learnable Evolution Model (LEM) [20] –in which an original machine learning method based on inductive rules is applied– although other approaches that apply decision trees [18] and hill climbing methods [2] can also be found. Examples on the use of classification paradigms in optimization for the continuous domain can also be found in the literature, such as the use of Gaussian modeling [1].

2.2 Main Steps of the EBCOA Approach

EBCOAs can be regarded as an Evolutionary Computation approach very similar to EDAs in the sense that there are steps such as the learning of a probabilistic graphical model and the posterior sampling of this model in order to obtain the individuals of the new population. These two steps are present in both paradigms, although the most relevant differences are precisely in the type of Bayesian network to build: in EDAs the learning algorithms applied are general purpose Bayesian network induction algorithms while EBCOAs build Bayesian classifiers using the information provided by the fitness function. In order to better compare the main differences between these two paradigms, Figs. 1 and 2 illustrate the EDA and EBCOA approaches respectively. These two figures evidence this difference, in which it appears clearly that the probabilistic graphical model learned in EBCOAs contains the additional class variable which is the parent of the rest, and is denoted as C. Note also the difference in both approaches regarding the selection of individuals –in EDAs– or the division in classes following a supervised classification approach made in EBCOAs. Figures 3 and 4 show the pseudocode of these two different paradigms for a clearer explanation of these main differences.

We denote by $\boldsymbol{X} = (X_1, \ldots, X_n)$ an n–dimensional random variable, and we represent by $\boldsymbol{x} = (x_1, \ldots, x_n)$ one of its possible instantiations –that is, one of the possible individuals. The probability distribution of $\boldsymbol{X}$ will be denoted as $p(\boldsymbol{x})$, and

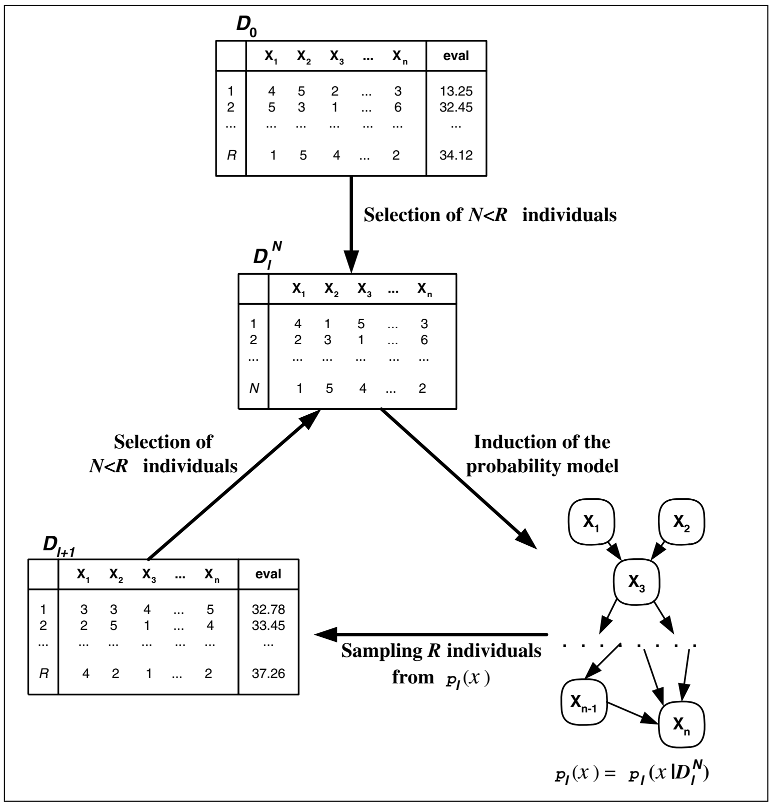

Fig. 1. Illustration of the EDA approach in the optimization process

the conditional probability distribution of the variable X_i given the value x_j of the variable X_j will be written as $p(x_i|x_j)$.

Let D_l be the population (database) of the l-th generation, formed by R individuals. This population has to evolve to the $(l+1)$-th one. In EBCOA, instead of having a selection of the fittest step as in EDAs, the population D_l is firstly divided in $|E|$ different classes, where following a supervised classification approach the variable E is defined so that to take the values $\{1, 2, \ldots, |E|\}$. We denote by D_l^E the database D_l after being divided in $|E|$ classes, in which each of the individuals in the population is assigned to a class of the variable E. In many cases we will be interested in enhancing the characteristics of the fittest and least fit classes, and therefore it is very likely not to use all the different classes for the learning. Therefore, we select $|C| \leq |E|$ classes and we ignore the rest of them for learning the Bayesian classifier. We denote by D_l^C the subset of D_l^E that will be used for the learning, and similarly we denote by C the variable that assigns a class c –with $1 \leq c \leq |C|$– to each of the individuals in D_l^C.

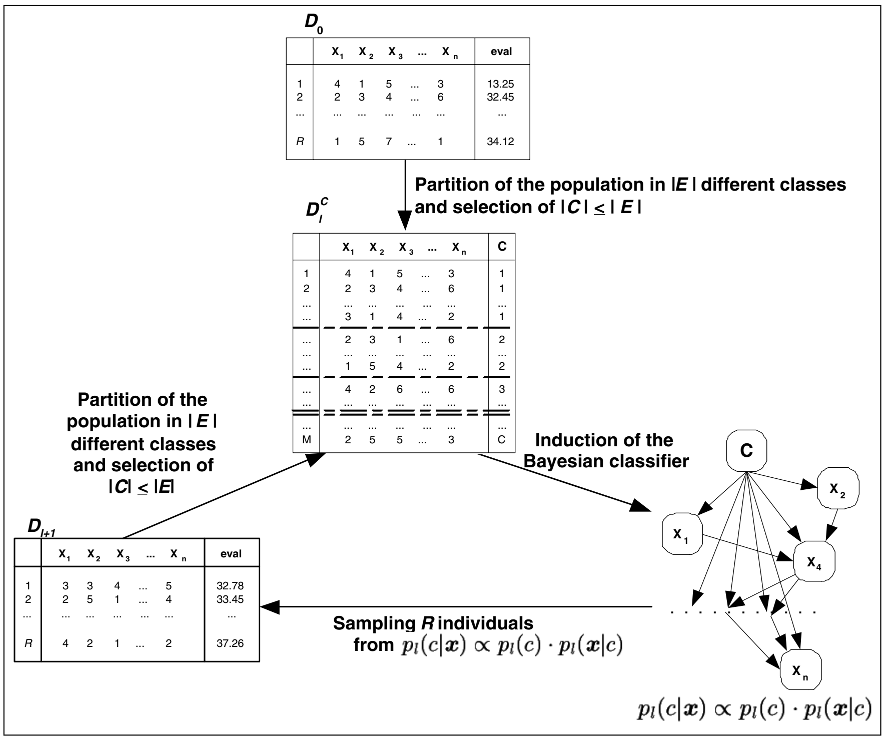
D_0
D_l^C
D_{l+1}
Partition of the population in |E| different classes and selection of |C| ≤ |E|
Partition of the population in |E| different classes and selection of |C| ≤ |E|
Induction of the Bayesian classifier
Sampling R individuals from $p_l(c|\boldsymbol{x}) \propto p_l(c) \cdot p_l(\boldsymbol{x}|c)$
$p_l(c|\boldsymbol{x}) \propto p_l(c) \cdot p_l(\boldsymbol{x}|c)$
C
X_1
X_2
X_4
X_n

Fig. 2. Illustration of the EBCOA approach in the optimization process

The hardest task in EBCOA and the most critical one regarding the convergence aspect, is the estimation of $p_l(\boldsymbol{x}, c)$. This probability is estimated from the Bayesian classifier that is learned every generation. As EBCOAs are based on Bayesian classifiers, the Bayesian network structure S that is induced as a result of the learning step contains the variables $X_1, \ldots, X_n$ as in EDAs, but also the newly defined variable C, and it will always be the parent of all the other variables in S. The next section introduces the main characteristics of the different methods for building Bayesian classifiers. See [22] for more details.

2.3 Bayesian Classifiers

The problem of *supervised classification* consists of assigning one of the $|C|$ classes of a variable C to a vector $\boldsymbol{x} = (x_1, \ldots, x_n)$. The true class is denoted by c and it takes values in $\{1, 2, \ldots, |C|\}$. Following this definition, a classifier can be seen as a function $\gamma : (x_1, \ldots, x_n) \rightarrow \{1, 2, \ldots, |C|\}$ that assigns a class label to each observation.

The loss function is defined as the cost of misclassifying an observation. A loss function $0/1$ is defined as a loss function in which the cost of misclassifying an element is always 1. In this case, it has been demonstrated that the optimum Bayesian

```
$D_0 \leftarrow$ Generate $R$ individuals (the initial population) at random
**Repeat** for $l = 0, 1, \ldots$ until satisfying a stopping criterion
    $D_l^N \leftarrow$ Select $N < R$ individuals of $D_l$ following
        a certain selection method
    $p_l(\boldsymbol{x}) = p(\boldsymbol{x}|D_l^N) \leftarrow$ Estimate the distribution of probability
        for an individual to be among the selected individuals
    $D_{l+1} \leftarrow$ Sample $R$ new individuals (the new population) from $p_l(\boldsymbol{x})$
```

Fig. 3. Generic pseudocode of EDA

```
$D_0 \leftarrow$ Generate $R$ individuals (the initial population) randomly
**Repeat** for $l = 0, 1, 2 \ldots$ until a stopping criterion is met
    $D_l^E \leftarrow$ Divide the $R$ individuals in $E < R$ different classes from $D_l$
        according to a criterion
    $D_l^C \leftarrow$ Select the $C \leq E$ classes of $D_l^E$ that will be used for building the
        Bayesian classifier, usually taking into account at least the best
        and worst classes.
        The individuals of the classes not included in $D_l^C \subset D_l^E$ are ignored
    $p_l(c|\boldsymbol{x}) \propto p_l(c) \cdot p_l(\boldsymbol{x}|c) \leftarrow$ Estimate the probability distribution of an individual
        in $D_l^C$ of being part of any of the different possible $C$ classes
    $D_{l+1} \leftarrow$ Sample $R$ individuals (the new population) from $p_l(\boldsymbol{x}|c)$
```

Fig. 4. Generic pseudocode for the EBCOA approach

classifier that minimizes the total misclassification error cost is obtained by assigning to the observation $\boldsymbol{x} = (x_1, \ldots, x_n)$ the class with the highest a posteriori probability [5].

This optimum Bayesian classifier is expressed more formally as follows:

$$\gamma(\boldsymbol{x}) = \arg\max_{c} p(c|\boldsymbol{x}) \tag{1}$$

In informative or generative classifiers such as the ones that are revised in this section, $p(c|x_1, \ldots, x_n)$ is obtained indirectly by applying the Bayes rule:

$$p(c|x_1, \ldots, x_n) \propto p(c, x_1, \ldots, x_n) \propto p(c)p(x_1, \ldots, x_n|c)$$

This section revises Bayesian classifiers that have been proposed specifically for classification problems.

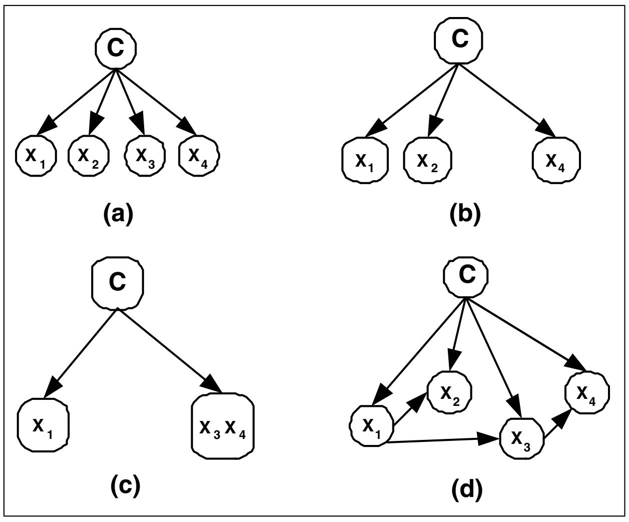

Fig. 5. Example of graphical structures of different Bayesian classifiers for a problem of four variables. The Bayesian classifiers presented are examples of (**a**) Naive Bayes, (**b**) Selective Naive Bayes, (**c**) Seminaive Bayes, and (**d**) Tree Augmented Naive Bayes

Naive Bayes

The naive Bayes approach [21] is the most simple one presented in this section. The Bayesian network structure that is applied is always fixed: all the variables $X_1, \ldots, X_n$ are considered to be conditionally independent given the value of the class value C. Figure 5(a) shows the structure that would be obtained in a problem with four variables.

In the naive Bayes classifier, when classifying an example $\boldsymbol{x}$, this will be assigned to the class c which has a higher a posteriori probability. This a posteriori probability is computed as follows:

$$p(c \mid \boldsymbol{x}) \propto p(c, \boldsymbol{x}) = p(c) \prod_{i=1}^{n} p(x_i|c) \tag{2}$$

The a priori probability of the class, $p(c)$, and the conditional probabilities $p(x_i|c)$ are estimated from the database of cases.

Selective Naive Bayes

The main restriction of naive Bayes is that this model forces the classifier to take into account all the variables. This aspect appears to be a drawback for some classification problems, since some of the observed variables could be irrelevant or redundant for classification purposes. Furthermore, it is known [13, 17] that the behavior of the naive Bayes paradigm degrades with redundant variables, and therefore the motivation for this approach is to remove those variables in order to obtain more efficient classifiers.

In the selective naive Bayes approach [14, 16] the variables in the classifier are considered to be independent as well as in naive Bayes, but in this case some of the variables can be ignored and not have them present in the final model. Figure 5(b) shows the structure that could be obtained in a problem with four variables, where one of them is missing in the final structure.

Following the selective naive Bayes model, and using the selective naive Bayes classifier shown in Fig. 5(a), an individual $\boldsymbol{x} = (x_1, x_2, x_3, x_4)$ would be assigned to the class

$$c^* = \arg\max_c p(c)p(x_1|c)p(x_2|c)p(x_4|c) \tag{3}$$

Seminaive Bayes

The previous two Bayesian classifiers have as a common property the fact that all the variables in the structure are considered to be conditionally independent. That is, all the variables can have uniquely the class variable C as a parent. The seminaive Bayes approach [15] is able to take into account dependencies between the variables $X_1, \cdots, X_n$ as it allows groups of variables to be considered as a single node in the Bayesian network. Figure 5(c) illustrates an example of a seminaive Bayesian classifier in a problem with four variables, showing that the Bayesian network structure treats those grouped variables as a single node regarding the factorization of the probability distribution. The grouping of variables as a single node means that all the dependencies between them are considered implicitly for classification purposes. On the other hand, and similarly as in selective naive Bayes, in seminaive Bayes it is also allowed that some variables are not included in the final classifier (Fig. 5(c) shows an example of this).

In [25] we can find a greedy approach to build seminaive Bayes classifiers, in which redundant as well as dependent variables are detected. When dependent variables are found, a new variable is created as the cartesian product of these. Two greedy algorithms are presented, the first of them on a forward direction called *FSSJ (Forward Sequential Selection and Joining)*, and the second on the opposite backward direction named *BSEJ (Backward Sequential Elimination and Joining)*. The pseudocode of *FSSJ* is shown in Fig. 6. The algorithm *BSEJ* follows an analogous approach, and can be interesting in optimization problems in which the objective function depends on all or nearly all the variables. Figure 5(c) shows an example of a possible structure for the Bayesian classifier that could be obtained as a result of the seminaive Bayes approach.

Therefore, applying the seminaive Bayes model and using the final classifier obtained in Fig. 5(c), an individual $\boldsymbol{x} = (x_1, x_2, x_3, x_4)$ would be assigned to the following class:

$$c^* = \arg\max_c p(c)p(x_1|c)p(x_3, x_4|c) \tag{4}$$

```
Initialize the set of variables to be used to the null set
Classify all the examples as being of the class with higher $p(c)$
**Repeat** in every iteration: choose the best option between
    (a) Consider each variable that is not in the model as a new one to be
        included in it. Each variable should be added as conditionally
        independent of the variables in the model given the class
    (b) Consider grouping each variable not present in the model with a variable
        that is already in it
    Evaluate each possible option by means of the estimation of the percentage
        of cases well classified
**Until** no improvement can be obtained
```

Fig. 6. Pseudocode of the *FSSJ* algorithm for seminaive Bayes models

Tree Augmented Naive Bayes

The tree augmented naive Bayes [7] is another Bayesian network classifier in which dependencies between the variables $X_1, \cdots, X_n$ are also taken into account, conditioned to the class variable C, by using a tree structure. In seminaive Bayes a wrapper approach[3] is applied to search for a good structure. In the tree augmented naive Bayes algorithm the method follows a procedure analogous to the filter approaches where only pairwise dependencies are considered.

The tree augmented naive Bayes structure is built in a two phase procedure illustrated in Fig. 7. Firstly, the dependencies between the different variables $X_1, \ldots, X_n$ are learned by means of a score based on the information theory. The weight of a branch (X_i, X_j) is given by $I(X_i, X_j|C)$, which is the mutual information measure conditioned to the class variable. These conditional mutual information values are used to build a tree structure. Secondly, the structure is augmented to the naive Bayes paradigm.

Following the tree augmented naive Bayes model building pseudocode presented in Fig. 7, if we obtain the Bayesian classifier structure shown in Fig. 5(d) an individual $\boldsymbol{x} = (x_1, x_2, x_3, x_4)$ will be assigned to the class

$$c^* = \arg\max_c p(c)p(x_1|c)p(x_2|c, x_1)p(x_3|c, x_1)p(x_4|c, x_3) \tag{5}$$

2.4 Description of the Main Steps of EBCOAs

Evolutionary Bayesian Classifier-based Optimization Algorithms (EBCOAs) is an approach that combines Bayesian classifiers such as the ones presented in the previous section and Evolutionary Computation to solve optimization problems. The

[3] The wrapper approach consists of applying the induction algorithm itself as a part of the evaluation function. On the other hand, the filter approach looks only at the intrinsic characteristics of the data, such as probabilistic or distance scores, or the mutual information

```
Calculate $I(X_i, X_j \mid C) = \sum_{i=1}^{n} \sum_{j=1}^{m} \sum_{r=1}^{w} p(x_i, y_j, c_r) \log \frac{p(x_i, y_j | c_r)}{p(x_i | c_r) p(y_j | c_r)}$
        with $i < j, j = 2, \ldots, n$
Build an undirected complete graph, where the nodes correspond to the predictor
    variables: $X_1, \ldots, X_n$. Assign the weight $I(X_i, X_j \mid C)$ to the edge connecting
    variables $X_i$ and $X_j$
Assign the largest two branches to the tree to be constructed

Repeat in every iteration:
    Examine the next largest branch and add it to the tree unless it forms a loop.
    In the latter case discard it and examine the next largest branch
Until $n - 1$ branches have been added to the structure

Transform the undirected graph in a directed one, by choosing a random
    variable as the root
Build the tree augmented naive Bayes structure adding a node labelled as $C$, and
    later add one arc from $C$ to each of the predictor variables $X_i$ $(i = 1, \ldots, n)$
```

Fig. 7. Pseudocode of the *tree augmented naive Bayes* algorithm

main idea is that, having a population of solutions for the optimization problem, this population will be evolved to a new generation formed by a next population of fitter individuals. The main difference between EDAs and EBCOAs is that in EDAs the evolution to the next population is performed by learning a probabilistic graphical model using uniquely the information of the best individuals, ignoring simply the worst ones, whilst in EBCOAs these worst individuals are also taken into account. The EBCOA approach is based on constructing a Bayesian classifier that will represent the main characteristics between the fittest and the least fit individuals. This approach, illustrated in Fig. 2, contains the following steps:

- Firstly, the initial population D_0 of R individuals is generated. This initial population is generated usually by assuming an uniform distribution on each variable, similarly as in EDAs. Each of the created individuals is evaluated by means of the fitness function.
- Secondly, each of the individuals in D_l are given a label $e \in E$ to classify them regarding their respective fitness value. This is the supervised classification step, in which each of the R individuals is assigned an e label. As a result of this, the class variable E is created in the new database denoted by D_l^E.
- Thirdly, D_l^C is created by selecting from D_l^E only the $|C| \leq |E|$ classes that will be used for building the Bayesian classifier. In EBCOAs we take into account uniquely the best and worst classes of individuals. The rest of the classes in D_l^E could be discarded to facilitate the learning by enhancing the differences between the most distant classes. The individuals which are in $D_l^E \backslash D_l^C$ are simply ignored.
- A Bayesian classifier is built using the database D_l^C and applying Bayesian classifier model construction techniques such as the ones described in the pre-

vious section. This classifier estimates the probability distribution $p_l(c|\boldsymbol{x}) \propto p_l(c)p_l(\boldsymbol{x}|c)$ which represents the probability of any individual $\boldsymbol{x}$ to be classified in any of the different possible $|C|$ classes.

- Finally, the new population D_{l+1} constituted by the R new individuals is obtained by carrying out the sampling of the probability distribution $p_l(c)p_l(\boldsymbol{x}|c)$. This step can be performed very similarly as in EDAs.

Steps 2, 3, 4 and 5 are repeated until a stopping criterion is satisfied. Examples of stopping conditions are: achieving a fixed number of populations or a fixed number of different evaluated individuals, uniformity in the generated population, and the fact of not obtaining an individual with a better fitness value after a certain number of generations.

The performance of EBCOAs is mainly responsibility of the step of learning the Bayesian classifier. In order to compare possible classifiers to be applied in EBCOAs, the most strict criterion to be used would be the use of an honest validation using the initial database of cases. However, this procedure is very expensive in computation time and very often this is approximated using a filter approach. It is important to note that in EBCOAs the most important criterion when choosing a Bayesian classifier is not to apply the one that best represents a strictly correct classifier, since the convergence speed and computation time are also important aspects to take into account. Indeed, the best Bayesian classifiers are usually the most time consuming ones. A balance between these two performance criteria is required since this learning step (i.e. the classifier building step) is going to be applied every generation.

3 The Satisfiability Problem

The SAT is one of the most known problems in computational theory because it results in a generic model in which many different decision making problems can be represented. The SAT problem is known to be NP-hard.

This particular problem has been analysed for many years and diverse methods have been applied for its resolution. Evolutionary computation methods have also been applied to it, mainly GAs, but also EDAs [11].

3.1 Definition of the Optimization Problem

The SAT problem is proposed in a formula in conjunctive normal form composed by a set of clauses. Each clause is formed by a set of literals, and a literal is a variable or its negation. A possible interpretation for the problem is a function that assigns a boolean value to each of the variables. An instantiation is said to be satisfiable if it satisfies every single clause. In order to satisfy a clause, it is necessary that at least one of the literals of the clause is satisfied by the instantiation.

The SAT problem is based on a set of Boolean variables $\mathbf{Z} = (Z_1, \ldots, Z_v)$, and a Boolean function $g : B^v \rightarrow B, B = \{0, 1\}$. The formula g is in conjunctive normal form $g(\mathbf{z}) = c_1(\mathbf{z}) \wedge \ldots \wedge c_u(\mathbf{z})$, where u is the number of clauses of the problem and each clause c_i is a disjunction of literals. A literal is a variable or its negation. A literal z_i is said to be satisfied if the value 1 is assigned to z_i, while the literal $\bar{z}_i$ is satisfied if the value 0 is assigned. A SAT instance is called *satisfiable* if there exists such an **z**, and otherwise it is called unsatisfiable. Here the aim is to analyze and check the existence of an assignation of $\mathbf{z} = (z_1, \ldots, z_v) \in B^v$ such that $g(\mathbf{z}) = 1$.

For example,

$$g(\mathbf{z}) = (z_1 \vee z_3 \vee \bar{z}_4) \wedge (z_2 \vee \bar{z}_3 \vee z_4) \wedge (\bar{z}_1 \vee z_2 \vee \bar{z}_3) \tag{6}$$

This instance is considered satisfiable if all its clauses are satisfied, and each of the single clauses is satisfiable if at least one of the literals is according to Z. For instance,

$$\mathbf{z} = (z_1, z_2, -, z_4) \tag{7}$$

satisfies the instance, where the value $-$ in Z_3 means that this variable can take the positive or negated value.

One might find easy to find a solution to this type of problems, but when we increase the number of variables and clauses the problem becomes very difficult to be solved.

The sub-type of SAT problems called *k*-SAT is defined as the one that contains in each clause exactly *k* different literals. While the *2*-SAT class is solved in polynomial time, *k*-SAT appears to be NP-hard for $k \geq 3$. In 1971, Cook [3] showed that this problem is NP-hard and that the SAT problem can be understood as a representative for solving other NP-hard problems.

The optimization version of the SAT problem is known as MAXSAT, and it consists on finding an assignation that maximizes the number of clauses satisfied for a particular SAT problem. If the original set of clauses is insatisfiable, MAXSAT is supposed to search for the assignation that satisfies the highest number of clauses possible.

3.2 Related Research Work

SAT is a generic method for problem solving that has been analysed for many years in the literature using very different approaches. This section concentrates on the work oriented to solve SAT by means of heuristic methods, and more precisely on Evolutionary Computation methods. In 1989, in [4] the authors apply GAs to the SAT problem to transform other NP-hard problems into SAT. This initial work concludes that GAs cannot perform better for the SAT, than other problem-specific algorithms, although GAs are referred to as a robust, efficient, and promising method.

More recent works apply different ways of representing solutions, techniques to adapt the definition of the fitness function, variations in crossover and mutation operators, or local optimization techniques. The aim of all these techniques in most of the cases is to improve the results obtained by GAs. As an example of these, [10] proposes to concentrate on the importance of the clause regarding the problem and proposes a new individual representation to guide the search following this idea.

In [9] this option and its corresponding fitness function with improvements is compared to using other type of representations, which also enhances the relevance of the clause, although the paper concentrates on the need to satisfy a single literal in each clause so that it becomes satisfied.

Finally, we would like to mention the work [8] in which the authors show an interesting revision of evolutionary algorithms for the SAT problem.

3.3 Representation Types and Their Associated Fitness Function

When solving any optimization problem using Evolutionary Computation techniques we need to choose a means of representing each of the possible solutions and to define a fitness function that measures how good each solution is for this problem. These two components are critical for an appropriated convergence of the algorithm. We can find in the literature many different options to represent a solution for the SAT problem. Each of these representations has a different appropriated fitness function. Next, we will analyse three different individual representations and their respective fitness functions: the first of them focuses on the individual literals that form the problem, while the other two are concentrated on the clauses and on how to satisfy them.

Bit-String Representation

This option is one the most applied in the literature, since it is the most natural way of representing the SAT problem. It consists on representing each possible solution as a string of bits of length v: $(z_0, \ldots, z_v)$, where v is the number of variables of the problem, and each variable Z_i is associated to a bit.

In this case, the aim is to satisfy the maximum number of clauses. Taking into account that the MAXSAT fitness function consists on counting the number of clauses that are satisfied, and the idea is to maximize this value, in the MAXSAT formulation the fitness value is equivalent to the number of satisfied clauses directly in the following way:

$$g_{MAXSAT}(\mathbf{z}) = c_1(\mathbf{z}) + \ldots + c_u(\mathbf{z}) \tag{8}$$

where c_i represents the true value of the $i\text{th}$ clause.

EXAMPLE 1:

Given Equation 6, a possible solution would be $\mathbf{z}_a = (1, 0, 0, 0)$ where $g_{MAXSAT}(\mathbf{z}_a) = 3$, since all the clauses are satisfied with this particular assignation.

On the other hand, if we consider another individual $\mathbf{z}_b = (1, 0, 1, 0)$, we have that $g_{MAXSAT}(\mathbf{z}_b) = 1$, since the second and third clauses are not satisfied.

Path Representation

This representation was suggested in [9]. It is based on the idea that for satisfying each particular clause it is enough with satisfying a single literal. Therefore, in order to create a possible satisfiable solution, it also would be possible to select a clause each time and generate a continuous path among clauses by choosing a literal in each single clause. That is, in this case a possible solution has as many variables as clauses, and each variable takes a value in $1, \ldots, k$ (k-SAT) which expressed the literal that is chosen in a particular clause.

The obvious problem created using this idea is that the same literal that appears in different clauses with different boolean values can be selected in two positions of the path, creating inconsistencies.

EXAMPLE 2:

Given Equation 6, the path (1-1-3), which means $z_1 = 1, z_2 = 1$, satisfies the SAT problem and induces the next complete assignations to literals: $\mathbf{z} = (1,1,0,0), \mathbf{z} = (1,1,0,1), \mathbf{z} = (1,1,1,0), \mathbf{z} = (1,1,1,1)$.

On the other hand, the path (1-3-1), $z_1 = 1, z_4 = 1, z_1 = 0$, contains an inconsistency for z_1.

A reasonable fitness function for this representation is the one that measures the number of inconsistencies, with the aim of minimizing this function.

If the path with a smaller number of inconsistencies has at least a single one, that particular SAT problem is not satisfiable. Otherwise, the fact of not having a single inconsistency in the path means that it exists at least a sequence of assigning variables that satisfies the SAT problem.

Clause Representation

A new individual representation also based on the clause-variables is proposed in [10], in which the effect of each of the variables within the clause is stressed. In other words, the main idea is to concentrate initially on finding an assignment of values to satisfy individual clauses, and next to search for the particular assignment of values in the literals of them to satisfy the problem globally.

Each clause with k literals can be regarded as to take $2^k - 1$ possible combinations of value assignments to its literals that satisfy the clause, while only one would not satisfy it. In the particular case of a *3*-SAT problem, there exist eight different possible values for each clause, and only one of would not satisfy the clause. Therefore, in this representation the main idea is to choose any of the $2^k - 1$ combinations of literal values in the clause that would satisfy it.

Following this idea, we can obtain an assignment that would satisfy all the clauses. However, this representation is likely to propose different values (according to whether the variable is negated or not) for the same literal which is present in different clauses. This case would result in an inconsistency in a similar way than in the previous path representation. Therefore, the idea is to choose any other clause-value so that the total number of inconsistencies among clauses is minimized.

Just as a simple example, for a clause $(z_2 \vee \bar{z}_3 \vee z_4)$, the assignation of variables not satisfied by this clause would be $(z_2, z_3, z_4) = (0, 1, 0)$. Next, we can see the illustration of how to use this representation in an overall 3-SAT problem.

EXAMPLE 3:
Given Equation 6 the forbidden assignations are: (0 0 1), (0 1 0),
(1 1 0) respectively, which correspond to the representation (1 2 6). A possible solution is (0 1 1), that is (0 0 0), (0 0 1), (0 0 1),
000 $\rightarrow z_1 = 0$, $z_3 = 0$, $z_4 = 0$,
001 $\rightarrow z_2 = 0$, $z_3 = 0$, $z_4 = 1$,
001 $\rightarrow z_1 = 0$, $z_3 = 0$, $z_2 = 1$,
however, this solution presents inconsistencies in z_2 and z_4.
On the other hand, the solution (2 6 3), that is (0 1 0), (1 1 0),
(0 1 1), does not present inconsistencies.
010 $\rightarrow z_1 = 0$, $z_3 = 1$, $z_4 = 0$,
110 $\rightarrow z_2 = 1$, $z_3 = 1$, $z_4 = 0$,
011 $\rightarrow z_1 = 0$, $z_3 = 1$, $z_2 = 1$,
therefore there exists a global assignation of variables (0 1 1 0) that satisfies the problem.

Similarly as with the previous representation, the fitness function proposed for this representation measures the amount of inconsistencies, and the aim of optimization algorithms is to minimize it.

When the clause representation is used, if the solution with the less possible number of inconsistencies has at least one inconsistency, the SAT is not satisfiable. Otherwise, the SAT problem is satisfiable.

3.4 Local Optimization

We propose in this section two types of local optimization techniques to converge to a satisfactory solution. The local optimization types presented in this section are applied in this case to EBCOAs, although they can also be used in other evolutionary computation methods. These techniques are suggested to be applied after the search process arrives to a determined generation number.

Firstly, we introduce a local optimization technique that flips each of the values in the individual after it has been generated, evaluating the fitness of each of the flips. If the individual is not improved, the original individual is included in the next generation, and otherwise, the fittest flipped version is included in the population. We call this method *local optimization with flip*.

Secondly, we present a method that detects the most difficult clauses of the problem to be satisfied, assigning them a higher weight in an adapted fitness function. This method is called *optimization through adaptation of weights*.

These local optimization procedures try to improve the search for a satisfiable version. Initially, the fitness function guides the search in a very efficient way, but due to the ambiguity on assigning the same fitness value to very different individuals that are given rise by many individual representations and fitness functions, this guidance is not precise enough to guide the algorithm towards finding a satisfiable solution in the search space. We suggest to apply this local optimization procedure once the search has found a number of different individuals that have the same best fitness value, since the algorithm will have no appropriated means to continue the search. The key decision here is to choose the best time to start applying the local optimization. The criterion that we propose is to start applying the local optimization when all the individuals of the highest (fittest) class have all the same fitness value. Other options are also possible, for instance to start to apply it once a concrete generation number has been reached.

Local Optimization with Flip

The local optimization that we propose in this section is based on the FlipGA method proposed in [19] for GAs. The main idea of FlipGA consists on applying the "Flip" heuristic to each individual after the crossover and mutation operations. The heuristic explores the genes randomly: each gen is "flipped", and this modification is accepted only if the improvement is bigger or equal than zero (that is, the number of satisfied clauses after the modification is the same or higher).

We propose in this section a procedure to apply local optimization as follows:

1. Each generated individual will be evaluated and its variables that worsen the individual – i.e. they make a clause not to be satisfied or create inconsistencies – will be marked.
2. Next, the marked variables are taken individually, and each time we modify a single variable and re-evaluate the individual. If a better fitness value is obtained, this modification is kept for this variable, otherwise we discard it.

Depending on the type of individual representation chosen, the local optimization will be applied in a different way.

Bit-string representation: The variables of the individuals represent directly the literals of the problem, while in the path and clause representations the variables of the individual represent the different clauses and the value of the literals are implicit.

Path representation: When a variable creates an inconsistency this means that for the same literal a clause has assigned a value while in another the value is the opposite. When this is identified, the literal that creates an inconsistency is marked in order to choose another literal of the clause and try to satisfy it. This helps at finding a satisfiable solution.

Clause representation: When a variable of the individual that creates an inconsistency, this means that the value of at least a literal that has been assigned to satisfy the clause is said to take a different value in two different clauses. This variable of the individual is marked and the local optimization will try to choose another combination of values of the clause to satisfy it and avoid the inconsistency.

Optimization Through Adaptation of Weights

The improvement presented in this section proposes to consider the different clauses differently according to the difficulty to satisfy each of them. This idea has been proposed for the first time in [6], and they apply the stepwise adaption of weights principle (SAW) combined with the fitness function.

$$g_{SAW}(\mathbf{z}) = w_1 \cdot c_1(\mathbf{z}) + \ldots + w_u \cdot c_u(\mathbf{z}) \tag{9}$$

The weights $w_i \in N$ are adapted to identify the most difficult clauses to satisfy in each step of the search. Initially, all the weights are initialized to 1 ($w_i = 1$), which is equivalent to apply a MAXSAT function. After some time, the weights are adapted according to $w_i \leftarrow w_i + 1 - q_i(z^*)$, where $q_i(z^*)$ is the actual fitness. This adaptation increases only the weights that correspond to clauses that have not been satisfied at that time. This forces implicitly the evolutionary search focusing it on these *difficult* clauses, and therefore guiding the search process applying these weights.

Here one of the tasks to perform is to decide when to start applying the optimization to the search. Similarly as in the previous case, we consider that the search process must start per se without applying the optimization procedure until some concrete conditions are detected. A possible proposal is to start applying this procedure when reaching a concrete generation number, or also when all the individuals in the fittest class have the same fitness value. Another decision to take is the number of generations after which we will adapt periodically the vector of weights.

4 Experimental Results

4.1 Comparison of the Different EBCOAs

In order to perform some experiments we have chosen *3*-SAT instances randomly from the collection of problems offered publicly by SATLIB [12]. The files selected contain definitions of different SAT problems for which we know that a satisfiable solution exists. We have used 5 different problem instances with 20 literals and 91 clauses each. For each instance we run each EBCOA presented in Sect. 2.3 (EBCOA$_{NaiveBayes}$, EBCOA$_{SelectiveNB}$, EBCOA$_{FSSJ}$, EBCOA$_{BSEJ}$ and EBCOA$_{TAN}$) 10 times under the same conditions. For each algorithm we have also applied three exposed possible individual representations and their respective fitness functions as described in Sect. 3.3.

Table 1 shows the results of these experiments for each individual representation type and EBCOA algorithm, without the addition of any local optimization at all. The column *Porc.* shows the percentage of runs in which a satisfiable solution has been found. The *Dif.* column represents the mean number of clauses that are not satisfied in the last generation of the runs. As we can see, this column corresponds to the percentage of runs in which a possible solution was found for the SAT problem: the value in *Dif.* is bigger when the percentage of runs in which a satisfiable solution

Table 1. Experimental results obtained with 5 different SAT problems containing 20 literals and 91 clauses. No local optimization technique has been applied in any of these results

	Bits			Path			Clause		
	Porc.	Dif.	Eval.	Porc.	Dif.	Eval.	Porc.	Dif.	Eval.
EBCOA$_{NaiveBayes}$	0.86	0.18	2222.84	0.06	1.72	54443.32	0.00	39.12	49832.62
EBCOA$_{SelectiveNB}$	0.82	0.20	2256.10	0.02	1.78	53788.66	0.00	41.78	49856.22
EBCOA$_{FSSJ}$	0.98	0.02	2472.10	0.00	2.76	50253.64	0.00	53.56	49753.52
EBCOA$_{BSEJ}$	0.92	0.08	1700.60	0.02	1.74	52330.42	0.00	49.50	49881.02
EBCOA$_{TAN}$	0.52	0.66	5285.12	0.00	4.60	52767.50	0.00	26.02	49639.46

has been found. On the other hand, the *Eval.* column represents the number of different individuals that have been evaluated, that is, the number of different individuals that have been analyzed in the search process. The stopping criterion chosen for all the cases is the fact of finding a satisfiable solution or to reach a maximum of 500 generations.

As we can see in Table 1 the bit-string representation is the one that shows the best performance. The other two representations obtain considerably worse results due to having a bigger search space and more different individuals with the same fitness value. The EBCOA that performed best is the one that learns a seminaive Bayes classifier, with which we obtain a satisfiable version in more than 90% of the executions. In the case of EBCOA$_{FSSJ}$ we obtain even a result of 98%. EBCOA$_{TAN}$ is the one that obtains a satisfiable solution in only 50% of the executions.

4.2 The Result of Applying Local Optimization

In order to analyse the effect of local optimization in EBCOAs, we have performed a set of experiments to compare the performance of not applying local optimization, applying the optimization described in Sect. 3.4, and the optimization through adaptation of weights of Sect. 3.4. Table 2 shows the results obtained in our experiments.

On the other hand, when local optimization with flip is applied more balanced results are obtained, and the differences between EBCOAs are reduced. This is the result on a better guidance for the different EBCOAs over the search space. EBCOA$_{TAN}$ improves its performance to the 76% of the executions, and EBCOA$_{FSSJ}$ obtains a lower percentage of success, although it is still far behind the 90%.

When the weight adaptation optimization is applied, a satisfiable solution is obtained in the 100% of the cases when applying an EBCOA based on a Selective Naive Bayes classifier. For the rest of EBCOAs the success rate is also quite high (98% for EBCOA$_{NaiveBayes}$ and EBCOA$_{FSSJ}$) except for EBCOA$_{TAN}$ which remains in a 72%.

It is also important to note that the effect of the individual representation is very important in EBCOAs. Other studies in the literature also show that the path representation has a worse performance than bit-string representation even when applied to GAs [9]; however, under the conditions of our experiments using EBCOAs the

Table 2. Experimental results obtained with 5 different SAT problems containing 20 literals and 91 clauses

Results applying local optimization with flip									
	Bits			Path			Clause		
	Porc.	Dif.	Eval.	Porc.	Dif.	Eval.	Porc.	Dif.	Eval.
EBCOA$_{NaiveBayes}$	0.92	0.08	1326.24	0.16	1.40	51113.16	0.00	12.52	49656.04
EBCOA$_{SelectiveNB}$	0.80	0.22	2274.98	0.12	1.48	50822.18	0.00	12.90	49654.14
EBCOA$_{FSSJ}$	0.92	0.08	2137.82	0.00	2.20	49956.84	0.00	29.28	49655.04
EBCOA$_{BSEJ}$	0.94	0.08	1806.74	0.06	2.02	48897.10	0.00	15.04	49600.00
EBCOA$_{TAN}$	0.76	0.28	3437.08	0.00	4.10	50130.92	0.00	70.44	49625.38

Results applying weight adaptation optimization									
	Bits			Path			Clause		
	Porc.	Dif.	Eval.	Porc.	Dif.	Eval.	Porc.	Dif.	Eval.
EBCOA$_{NaiveBayes}$	0.98	0.04	2155.10	0.58	0.76	31726.54	0.00	12.58	49600.00
EBCOA$_{SelectiveNB}$	1.00	0.00	4912.80	0.52	0.88	34284.70	0.00	11.42	49600.00
EBCOA$_{FSSJ}$	0.98	0.04	3105.50	0.00	5.86	49600.00	0.00	33.86	49600.00
EBCOA$_{BSEJ}$	0.94	0.08	5420.42	0.64	0.76	32599.72	0.00	12.38	49600.00
EBCOA$_{TAN}$	0.72	0.38	14698.92	0.00	6.42	49600.00	0.00	47.86	49600.00

performance is quite poor. EBCOAs are able to find a satisfiable version when using this representation, and only if we apply weight adaptation optimization we manage to find satisfiable solutions in more than 50% of the cases – but not for all the EBCOA types. Results are even worse for the case of clause representation, since no EBCOA did manage to find a satisfiable solution in any of the executions.

4.3 Performance of EBCOAs Depending on the Complexity of the SAT Problem

It is a common practise to define the complexity of a SAT problem by using a u/v ratio, where u is the number of clauses and v is the number of variables. In all our experiments, we use instances with ratio $u/v \approx 4,3$, which have been reported by [23] to be the hardest instances. With this ratio, we performed an experiment to analyse the performance of EBCOAs when increasing the number of literals of the SAT problem. For this, we have applied other problems from SATLIB [12] containing SAT with 50 literals and 218 clauses. We chose again 5 different instances of these SAT problems and we run 10 times each EBCOA using the different local optimization possibilities. This time we have applied uniquely the bit-string representation. The results obtained are presented in Table 3.

As we can see in this table, the tendency shown in the previous experiments is kept for this case, although the increase in complexity results in a much lower performance than the case with 20 literals.

Table 3. Results for a SAT problem with 50 literals and 218 clauses. These results have been obtained using a bit-string representation

Results without local optimization

	Porc.	Dif.	Eval.
EBCOA$_{NaiveBayes}$	0.18	1.62	15933.72
EBCOA$_{SelectiveNB}$	0.08	1.78	16633.78
EBCOA$_{FSSJ}$	0.34	1.14	98698.86
EBCOA$_{BSEJ}$	0.14	1.90	17962.96
EBCOA$_{TAN}$	0.00	4.06	25691.80

Results applying local optimization with flip

	Porc.	Dif.	Eval.
EBCOA$_{NaiveBayes}$	0.10	1.62	13620.32
EBCOA$_{SelectiveNB}$	0.20	1.36	14232.72
EBCOA$_{FSSJ}$	0.46	0.94	90296.14
EBCOA$_{BSEJ}$	0.26	1.44	14749.36
EBCOA$_{TAN}$	0.00	3.76	20471.18

Results applying weight adaptation optimization

	Porc.	Dif.	Eval.
EBCOA$_{NaiveBayes}$	0.50	1.06	69199.38
EBCOA$_{SelectiveNB}$	0.52	0.94	66921.32
EBCOA$_{FSSJ}$	0.62	0.74	72924.72
EBCOA$_{BSEJ}$	0.46	1.14	71453.36
EBCOA$_{TAN}$	0.10	2.82	110144.16

5 Conclusions

This chapter presents a new evolutionary computation approach called EBCOA, and its performance has been studied for a classical NP-hard problem, the satisfiability one. We have analysed the behavior of EBCOAs for three different individual representations together with the appropriated fitness function for each case. In our experimental results we demonstrate that the bit-string representation obtains the best results than the ones based on the clause structures. This is due to the fact that the bit-string representation reduces the search space since the variables of the individual are binary.

We have also applied two different local optimization options based on the specific characteristics of the SAT problem. In this sense, in one of them we differentiate the clauses which are more difficult to be satisfied and a weight is assigned to each clause regarding this aspect. The results obtained in this local optimization method are the most promising.

EBCOAs are a very new research paradigm that has still a lot of work to be done. Possible future working trends that we are currently facing are the use of other more complex Bayesian classifiers, the definition of other mechanisms to adapt the learning step in EBCOAs to take into account the a priori information of the optimization problem, and the use of other local optimization techniques designed specifically for EBCOAs.

Acknowledgements

This chapter has been partially supported by the University of the Basque Country with the project 9/UPV-EHU 00140.226-15334/2003, as well as by the Basque Government with projects ETORTEK BIOLAN, and ETORTEK GEMMODIS. The authors would like to thank these institutions for their kind support.

References

1. S. Akaho. Statistical learning in optimization: Gaussian modeling for population search. In *Proc. of 5th International Conference on Neural Information Processing*, pp. 675–678, 1998.
2. J.A. Boyan and A.W. Moore. Learning evaluation functions for global optimization and boolean satisfiability. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence AAAI-98*, 1998.
3. S. Cook. The complexity of theorem-proving procedures. In J.J. Grefenstette, editor, *In Proceedings of Third Annual ACM Symposium on Theory of Computing*, pp. 151–158, New York, 1971.
4. K. de Jong and W. Spears. Using genetic algorithms to solve NP-complete problems. In J.D. Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms*, pp. 124–132, San Mateo, CA, 1989. Morgan Kaufmann.
5. R. Duda and P. Hart. *Pattern Classification and Scene Analysis*. John Wiley and Sons, New York, 1973.
6. A.E. Eiben and J.K. van der Hauw. Solving 3-SAT by GAs adapting constraint weights. In *Proceedings of The IEEE Conference on Evolutionary Computation, IEEE World Congress on Computational Intelligence*, 1997.
7. N. Friedman, D. Geiger, and M. Goldsmidt. Bayesian network classifiers. *Machine Learning*, 29(2):131–163, 1997.
8. J. Gottlieb, E. Marchiori, and C. Rossi. Evolutionary Algorithms for the Satisfiability Problem. *Evolutionary Computation*, 10(1):35–50, 2002.
9. J. Gottlieb and N. Voss. Representations, fitness functions and genetic operators for the satisfiability problem. In *Proceedings of Artificial Evolution. Lecture Notes in Computer Science*, volume 1363, pp. 55–68, Springer, Berlin, Germany, 1998.
10. J.-K. Hao. A clausal genetic representation and its evolutionary procedures for satisfiability problems. In D. Pearson et al., editor, *Proceedings of the International Conference on Artificial Neural Nets and Genetic Algorithms*, pp. 289–292, Vienna, Austria, 1995. Springer.
11. V. Herves, P. Larrañaga, V. Robles, J. M. Peña, M. S. Pérez, and F. Rosales. EDA paralelos multipoblación para el problema SAT. In *Proceedings of the III Congreso Español de Metaheurísticas, Algoritmos Evolutivos y Bioinspirados*, Cordoba (Spain), 2004.
12. H.H. Hoos and T. Stützle. SATLIB: An online resource for research on SAT. In I.P.Gent et al., editor, *SAT 2000*, pp. 283–292. IOS Press, 2000. SATLIB is available online at www.satlib.org.

13. I. Inza, P. Larrañaga, R. Etxeberria, and B. Sierra. Feature subset selection by Bayesian network-based optimization. *Artificial Intelligence*, 123(1-2):157–184, 2000.
14. R. Kohavi and G. John. Wrappers for feature subset selection. *Artificial Intelligence*, 97(1-2):273–324, 1997.
15. I. Kononenko. Semi-naïve Bayesian classifiers. In *Proceedings of the 6th European Working Session on Learning*, pp. 206–219, Porto, Portugal, 1991.
16. P. Langley and S. Sage. Induction of selective Bayesian classifiers. In *Proceedings of the 10th Conference on Uncertainty in Artificial Intelligence*, pp. 399–406, Seattle, WA, 1994.
17. H. Liu and H. Motoda. *Feature Selection for Knowledge Discovery and Data Mining*. Kluwer Academic Publishers, Boston, 1998.
18. X. Llorà and D.E. Goldberg. Wise breeding GA via machine learning techniques for function optimization. In Cantú-Paz et al., editor, *Proceedings of the Genetic and Evolutionary Computation Conference GECCO-03, Part I*, Lecture Notes in Computer Science 2723, pp. 1172–1183, Chicago, Illinois, 2003. Springer.
19. E. Marchiori and C. Rossi. A flipping genetic algorithm for hard 3-SAT problems. In W. Banzhaf et al., editor, *Proceedings of Genetic and Evolutionary Computation Conference*, pp. 393–400, San Francisco, CA, 1999. Morgan Kaufmann.
20. R.S. Michalski. Learnable evolution model: Evolutionary processes guided by machine learning. *Machine Learning*, 38:9–40, 2000.
21. M. Minsky. Steps toward artificial intelligence. *Transactions on Institute of Radio Engineers*, 49:8–30, 1961.
22. T. Miquélez, E. Bengoetxea, and P. Larrañaga. Evolutionary computation based on Bayesian classifiers. *International Journal of Applied Mathematics and Computer Science*, 14(3):335–349, 2004.
23. D. G. Mitchell, B. Selman, and H. J. Levesque. Hard and easy distributions for SAT problems. In P. Rosenbloom and P. Szolovits, editors, *Proceedings of the Tenth National Conference on Artificial Intelligence*, pp. 459–465, Menlo Park, California, 1992. AAAI Press.
24. H. Mühlenbein and T. Mahning. FDA - a scalable evolutionary algorithm for the optimization of additively decomposed functions. *Evolutionary Computation*, 7(4):353–376, 1999.
25. M. Pazzani. Searching for dependencies in Bayesian classifiers. In D. Fisher and H.-J. Lenz, editors, *Learning from Data: Artificial Intelligence and Statistics V*, pp. 239–248, New York, NY, 1997. Springer–Verlag.
26. G. Syswerda. Simulated crossover in genetic algorithms. In L.D. Whitley, editor, *Foundations of Genetic Algorithms*, volume 2, pp. 239–255, San Mateo, California, 1993. Morgan Kaufmann.