[7] C. D. Rosin and R. K. Belew, "New methods for competitiev coevolution," *Evolut. Comput.*, vol. 5, no. 1, pp. 1–29, 1997.
[8] M. A. Potter, K. D. Jong, and J. Grefenstette, "A coevolutionary approach to learning sequential decision rules," in *Proc. 6th Int. Conf. Genetic Algorithms*, Pittsburgh, PA, 1995.
[9] C. A. Peña-Reyes and M. Sipper, "Fuzzy CoCo: A cooperative-coevolutionary approach to fuzzy modeling," *IEEE Trans. Fuzzy Syst.*, vol. 9, no. 5, pp. 727–737, Oct. 2001.
[10] D. E. Moriarty and R. Miikkulainen, "Efficient reinforcement learning through symbiotic evolution," *Mach. Learn.*, vol. 22, pp. 11–32, 1996.
[11] ——, "Hierarchical evolution of neural networks," in *Proc. IEEE Conf. Evoulutionary Computation*, Anchorage, AK, 1998, pp. 428–433.
[12] C. F. Juang, J. Y. Lin, and C. T. Lin, "Genetic reinforcement learning through symbiotic evolution for fuzzy controller design," *IEEE Trans. Syst., Man, Cybern. B, Cybern.*, vol. 30, no. 2, pp. 290–302, Apr. 2000.
[13] J. L. Elman, "Finding structure in time," *Cogn. Sci.*, vol. 14, pp. 179–211, 1990.

Knowledge-Based Fast Evaluation for Evolutionary Learning

Raúl Giráldez, Jesús S. Aguilar-Ruiz, and José C. Riquelme

***Abstract*—The increasing amount of information available is encouraging the search for efficient techniques to improve the data mining methods, especially those which consume great computational resources, such as evolutionary computation. Efficacy and efficiency are two critical aspects for knowledge-based techniques. The incorporation of knowledge into evolutionary algorithms (EAs) should provide either better solutions (efficacy) or the equivalent solutions in shorter time (efficiency), regarding the same evolutionary algorithm without incorporating such knowledge. In this paper, we categorize and summarize some of the incorporation of knowledge techniques for evolutionary algorithms and present a novel data structure, called efficient evaluation structure (EES), which helps the evolutionary algorithm to provide decision rules using less computational resources. The EES-based EA is tested and compared to another EA system and the experimental results show the quality of our approach, reducing the computational cost about 50%, maintaining the global accuracy of the final set of decision rules.**

***Index Terms*—Data structures, evolutionary algorithms (EAs), knowledge incorporation, supervised learning.**

I. Introduction

Evolutionary computation techniques are commonly used to address problems with large search spaces, where an exhaustive search is not applicable in practice. Due to this fact, many machine learning and optimization tasks have been solved by using genetic algorithms (GAs) or evolutionary algorithms (EAs) [12]. Incorporating some domain knowledge to evolutionary algorithms at initial stages will improve their performance by driving individuals to better solutions. Nevertheless, some authors have focused their researches on how to add knowledge during the evolutionary process in order to improve its performance in two directions: *efficacy*, by biasing the search toward better solutions (without increasing the computational cost), and *efficiency*, by reducing the computational cost of the algorithms (maintaining the quality of solutions).

Manuscript received September 1, 2003; revised February 2, 2004 and April 21, 2004. This work was supported by the Spanish Research Agency CICYT under Grant TIN2004-00159. This paper was recommended by Guest Editor Y. Jin.

The authors are with the Department of Computer Science, University of Seville, Seville 41012, Spain (e-mail: giraldez@lsi.us.es; aguilar@lsi.us.es; riquelme@lsi.us.es).

Digital Object Identifier 10.1109/TSMCC.2004.841904

The EA should generate solutions with better quality when knowledge incorporation is used [18]. If the problem belongs to the optimization field, precision is preferred. However, if it is a supervised learning problem, the size and complexity of the knowledge model are taken into account as well, as these are not only used to predict new unseen data, but also to give insight about the relevance of features to understand the phenomenon.

In this paper we categorize and briefly summarize some of the incorporation of knowledge techniques for evolutionary algorithms and present a novel data structure, called efficient evaluation structure (EES), which helps the evolutionary algorithm to provide decision rules using less computational resources. Experimental results show the quality of our proposal, reducing the computational cost about 50%, maintaining the global accuracy of the final set of decision rules.

The paper is organized as follows. In Section II, the incorporation of knowledge techniques in EAs are categorized and briefly summarized. Section III describes important aspects of the evolutionary learning of decision rules and differences between linear and EES-based evaluation; the EEES is presented in Section IV. Experimental results are shown in Section V, and in Section VI the most interesting conclusions are summarized.

II. Related Work

The EAs that incorporate domain knowledge can be classified in two groups depending on when the knowledge is extracted. The first group brings together those techniques that extract the domain knowledge before starting the search for solutions. We name this group *evolutionary algorithms with a priori knowledge extraction* (EAAPs). The extracted knowledge is therefore constant along the entire evolutionary process. The second group, *evolutionary algorithms with dynamic knowledge extraction* (EADs) extract and incorporate the knowledge during the evolutionary process. In this case, the knowledge is updated throughout evolution.

On the other hand, EAs can also be divided into two categories depending on when the extracted knowledge is incorporated: *adapted EAs* and *adaptive EAs* (Fig. 1). The adapted EAs integrate domain knowledge before applying the evolutionary process (i.e., during the encoding design and population initialization phases). On the opposite, the adaptive EAs incorporate such knowledge during the genetic evolution. Thus, these algorithms evolve a dynamic adaptation of the parameters, the evaluation or the genetic operators. In next subsections, some strategies followed by adapted and adaptive EAs according to the phase where they are applied are discussed. As EADs extract the knowledge during evolution, they cannot be *adapted* but *adaptive*, since the knowledge has to be incorporated during the evolutionary process.

A. Adapted EAs

1) Coding: Some tools incorporate knowledge in the genetic representation. In this sense, Louis and Zhao [23] proposed a coding that improved the performance of an EA applied to the optimization tasks in the industrial design problems; specifically, they integrated engineering domain knowledge into the algorithm to synthesize topology, geometry, and component properties of complex structures. In the area of supervised learning, the system HIDER uses a particular evolutionary representation named Natural Coding [11]. This encoding reduces the search space by applying a supervised discretization method called USD [10], which maximizes the global accuracy of the intervals and does not use any user-defined parameter. Each gene in the individual

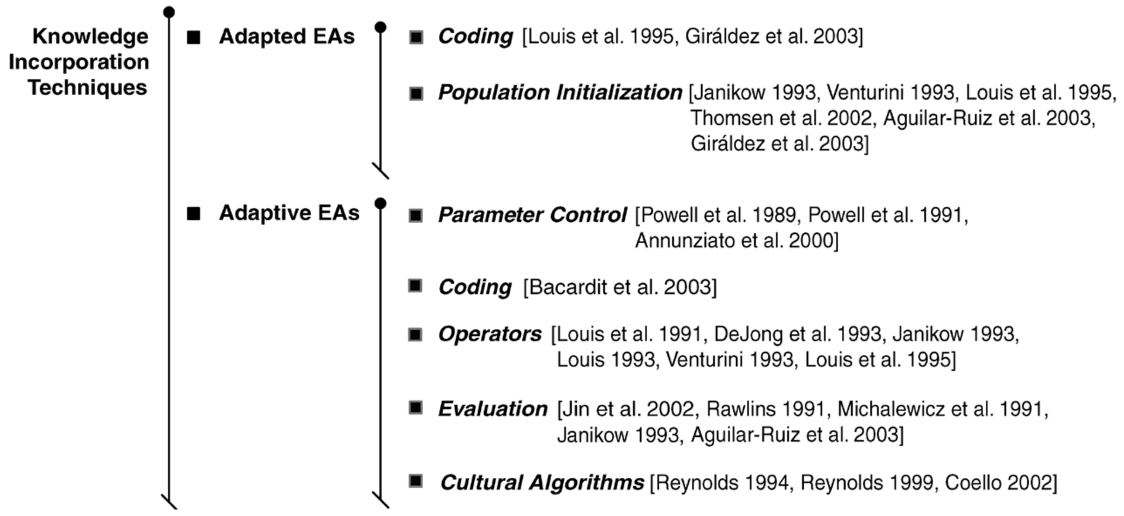

Knowledge Incorporation Techniques
Adapted EAs
Coding [Louis et al. 1995, Giráldez et al. 2003]
Population Initialization [Janikow 1993, Venturini 1993, Louis et al. 1995, Thomsen et al. 2002, Aguilar-Ruiz et al. 2003, Giráldez et al. 2003]
Adaptive EAs
Parameter Control [Powell et al. 1989, Powell et al. 1991, Annunziato et al. 2000]
Coding [Bacardit et al. 2003]
Operators [Louis et al. 1991, DeJong et al. 1993, Janikow 1993, Louis 1993, Venturini 1993, Louis et al. 1995]
Evaluation [Jin et al. 2002, Rawlins 1991, Michalewicz et al. 1991, Janikow 1993, Aguilar-Ruiz et al. 2003]
Cultural Algorithms [Reynolds 1994, Reynolds 1999, Coello 2002]

Fig. 1. Categorization of knowledge incorporation into evolutionary algorithms techniques.

is related to only one attribute, so specific genetic operators were designed for this encoding. The Natural Coding is oriented to supervised learning problems.

2) Population Initialization: In general, EAs can create the initial population on a number of ways. The simplest way is the random initialization, where the individuals are randomly generated. Some methods apply a grid initialization using a uniform distribution. Despite the fact that these are the most common setups, they do not incorporate knowledge to the initial population. Since domain experts usually have an idea of what results are reasonable, an early approach lies in using the expert knowledge in order to locate the initial search space positions. Thus, the individuals could then be scattered near such positions by making use of the random or grid distribution. However, a randomly initialized population might mainly allow the EA to discover different solutions in comparison to those a human would have proposed. The most advanced initialization techniques includes potential solutions into the initial population that were created by other previously applied search techniques. The potential of this methods was investigated by Thomsen *et al.* in [38], and their approach reduced the running time of the EA significantly. This could be named "wrapper initialization", as hypothetical individuals are evaluated by other methods, and those that present better fitness are selected to take part in the initial population.

In the particular case of EA-based Learning, some EAs select instances and generate solutions for them as initial individuals in order to preserve the diversity of starting points. These initial individuals are generated by varying slightly randomly chosen examples from the dataset. Several EAs with this initialization method can be found in the Machine Learning literature, (e.g., SIA [39], GIL [17] and HIDER [1]). A simple adaptation of the aforementioned Thomsen's wrapper initialization to supervised learning could be based on decision trees: one generates the decision tree by using See5 [30] and select all the paths from the root to leaves as individuals for the initial population. In the worst case, the individuals will not be improved at the end of the evolutionary process, therefore achieving the same accuracy as the See5 tool.

B. Adaptive EAs

1) Parameter Control: Running an EA entails setting a number of parameter values (population size, number of generations and genetic operators probabilities). This is a nontrivial task that can have severe influence on the algorithm performance. Some researchers have approached the problem from the perspective of self-adaptability, adding domain knowledge to the parameterization [3]. Among the first approaches, we can point out the algorithm named EnGENEuos, developed by Powell *et al.* [28], which combines a GA with an expert system that calculates the parameter values during the course of the evolutionary process. This hybrid system had good results, but it was very inefficient at certain applications. In order to solve this problem, Powell proposed later the "Interdigitation" method [29], integrating numerical optimization techniques.

2) Coding: In some cases the aim is to refine the representation during the evolutionary process. In the field of Evolutionary Learning, when the problem is related to classification, decision rules contains intervals to which attribute values might belong. Several approaches have been addressed with success, as evolving multiple discretizations with adaptive intervals. GAssist [4] is a Pittsburgh Genetic-Based Machine Learning system descendant of GABIL [7] and it evolves individuals that are ordered variable-length rule sets. The discretization intervals can evolve through the learning process splitting or merging among them. The representation can also combine several discretizations at the same time, allowing the system to choose the correct discretizer for each attribute.

3) Operators: Knowledge-based genetic operators have been satisfactorily applied in a number of EAs (e.g., Louis' works in the context of the structural design and optimization of trusses [21]–[23]). In the area of machine learning, systems like GIL [17], GABIL [7] and SIA [39] have incorporated knowledge-based operators into the EAs in order to improve the generalization and specialization of solutions.

4) Evaluation: In principle, it is possible to compare two solutions and decide which is the best according to a measurement of the solution quality that an expert could give [31]. Nevertheless, there are other ways for incorporating knowledge to the individuals evaluation process and for obtaining nonstatic fitness functions. For example, in [24] is presented an approach that incorporates constraint handling techniques to the evaluation. In [36] a new adaptive penalty approach is proposed for solving constrained optimization problems. Other techniques add a learning method to the EA that learns the fitness function as the evolutionary process advances. Fitness function estimation is gathering attention among researchers, as databases are growing quickly. Interesting fitness approximation techniques can be found in [19], [20], mainly for optimization problems. This idea has also been adapted to learning tasks. In this sense, the Neuro-Evolutionary Model (NEM [2]) includes a neural network which is trained to estimate the fitness function during the evolutionary process, so the EA gathers speed in time. Other simpler approaches, although also efficient, was applied in the context of Machine Learning (e.g., Janikow's concept learner (GIL [17]) uses a dynamic fitness function that depends on the population age).

$$\textbf{If } Cond_1 \textbf{ and } Cond_2 \textbf{ and} \ldots \textbf{and } Cond_M \textbf{ then } Class = C$$

Fig. 2. Decision Rule.

5) Cultural Algorithms: Cultural Algorithms (CA) [32], [33] do not carry out an *a priori* extraction of the knowledge but during the evolutionary process itself. CAs are bio-inspired algorithms that implement beyond natural selection. Besides the information than an individual inherits from its parents through the genetic code, there is another influential aspect called *culture*. Culture is a library of experiences where individuals stock the information acquired after years. Thus, new individuals can use these experiences to improve their evolution even when they have not learnt them directly [6].

III. Evolutionary Learning of Decision Rules

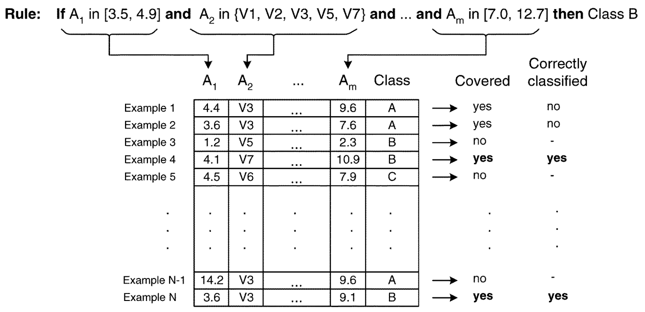
This work is focused on knowledge-based evaluation in the field of evolutionary learning, i.e., on the evaluation of individuals from the populations by using some strategy that incorporates knowledge into the evaluation process.

The knowledge model can be represented by different structures: decision rules, association rules, decision trees, etc. In Supervised Learning, the structures generated are usually decision rules or trees. A decision rule establishes conditions of which the examples must fulfil in order to be classified by this rule. These conditions affect the values that the attributes can take in that if the attributes of an example meet all the conditions that a rule establishes then we say that the rule covers the example, independent of whether it classifies such example correctly or not. The usual representation of rules generated by learning systems is shown in Fig. 2, where Cond_i is the condition that the ith attribute of an example has to satisfy in order to be classified with class C, and M is the number of attributes in the dataset. Logically, the case can arise where an attribute does not appear in the rule, from which we assume that the condition concerning the attribute is always evaluated as *true*. When an attribute (a_i) is continuous, the condition (Cond_i) takes the form $a_i \in [l_i, u_i]$, restricting the range of values of the attribute to the interval defined by the lower (l_i) and the upper (u_i) bound. On the other hand, when the attribute is discrete, the condition takes the form $a_i \in \{v_1, v_2, \ldots, v_k\}$, where the values $\{v_1, v_2, \ldots, v_k\}$ are not necessarily all those the attribute can take.

The EA-based learning methods usually evaluate the rules directly from the dataset. They explore such dataset sequentially, taking each one of the examples and testing the quality of the rules through the correct classification of those examples. We can see, therefore, that the learning process of these systems is very costly in terms of time and space. Some authors [30], [35] have concentrated their efforts on improving the learning process by speeding up the algorithm, in order to reduce its computational cost. Others have approached the problem from the perspective of scalability [37]. However, the appropriate organization of the information could also contribute to the reduction of the time complexity.

To find these rules many different techniques could be applied, EAs among them. Two critical factors have influence on the quality of the decision rules obtained by an EA: the selection of an internal representation of the search space (encoding), which determines the genetic operators; and the definition of an external function that assigns a value of godness to the potencial solutions (evaluation). In [1], a hybrid encoding is introduced, in which continuous attributes are represented by two real values and discrete attributes by a set of binary values (one means that the discrete value is active, and zero that it is not).

Our approach to incorporate knowledge to the EA is oriented to improve the two aforementioned factors. On the one hand, the data structure conditions the individual encoding and the mutation and crossover operators, what leads to find better solutions. On the other hand, the fitness evaluation is more efficient.



Rule: **If** A_1 in [3.5, 4.9] **and** A_2 in {V1, V2, V3, V5, V7} **and** ... **and** A_m in [7.0, 12.7] **then** Class B

	A_1	A_2	...	A_m	Class		Covered	Correctly classified
Example 1	4.4	V3	...	9.6	A	$\longrightarrow$	yes	no
Example 2	3.6	V3	...	7.6	A	$\longrightarrow$	yes	no
Example 3	1.2	V5	...	2.3	B	$\longrightarrow$	no	-
Example 4	4.1	V7	...	10.9	B	$\longrightarrow$	**yes**	**yes**
Example 5	4.5	V6	...	7.9	C	$\longrightarrow$	no	-
⋮	⋮	⋮	⋮	⋮	⋮		⋮	⋮
Example N-1	14.2	V3	...	9.6	A	$\longrightarrow$	no	-
Example N	3.6	V3	...	9.1	B	$\longrightarrow$	**yes**	**yes**

Fig. 3. Evaluation of individuals using a vector of examples.

The aim of our research is to design a data structure that incorporates knowledge on the example distribution in the attribute space. This information is very useful to locate the regions that must be explored in such space, to evaluate the individuals only in those regions, and thus to speed up the evaluation process.

Our EES organizes the information from a dataset in such a way that it is not necessary to process all the examples to evaluate decision rules generated by a supervised learning system.

A. Data Structures

There exist in the literature a number of approaches on data structures and organization of the information which essentially speed up the access to information in multidimensional spaces, such as those known as Multidimensional Access Methods (MAM) [9]: Point Access Methods (Grid File [26], KDB-tree [34], LSD-tree [14], BV-tree [8], etc.) and Spatial Access Methods (K-D-Tree [5], R-tree [13], P-tree [16], SDK-tree [27], etc.). However, given the peculiarity of the problem we deal with, MAMs do not, in themselves, provide a solution to such problem, since they index a dataset with the only goal to speed up queries on such data. For instance, AD-Tree [25] is the MAM that gives the nearest solution to the problem we want to solve. Nevertheless, if we wanted to evaluate decision rules using this structure, we would have to build the nonsparse AD-Tree, that is, to store all possible queries in addition to the indices of the examples that each query includes. This means very high computational cost, since too much redundant information is stored.

B. Linear Evaluation

An EA encodes candidate decision rules as individuals in the genetic population. Once the initial population is built the individuals are evaluated and depending on the quality of each one of them, the crossover and mutation operators are applied, thus constructing the next generation. It is simple to see that those systems that apply EAs need to carry out a constant number of evaluations throughout the learning process, since they have to evaluate each one of the individuals for each one of the populations that they generate. For example, an EA with 300 generations, each containing 100 individuals, needs to calculate 30 000 fitness evaluations. If the dataset contains 1 000 examples then it will be 30 million of evaluations. In the particular case of EAs applied to the generation of rules, the dataset is usually stored in a vector of examples (see Fig. 3), which is used for evaluating every individual. Not all the covered examples might be correctly classified by the rule encoded in the invididual (in Fig. 3, only the 4th and Nth examples are correctly classified, although the first and third examples are also covered by the rule, but the class is different). This information is very important to evaluate the individuals, and it should be interesting to know this without checking every example. However, if the evolutionary system uses directly the vector of examples, we need to check every example from the dataset.

Therefore the computational cost of a single evaluation is $\mathrm{O}(NM)$, where N is the number of examples and M is the number of attributes

N	A_1:Size	A_2:Colour	Class
1	2.0	blue	A
2	1.0	green	A
3	1.2	green	A
4	1.4	blue	A
5	1.0	red	A
6	1.4	red	A
7	1.7	blue	B
8	1.8	green	B
9	1.2	red	B
10	2.1	blue	A
11	2.2	green	A
12	1.0	blue	A
13	1.2	blue	B
14	2.0	green	A
15	1.6	blue	B

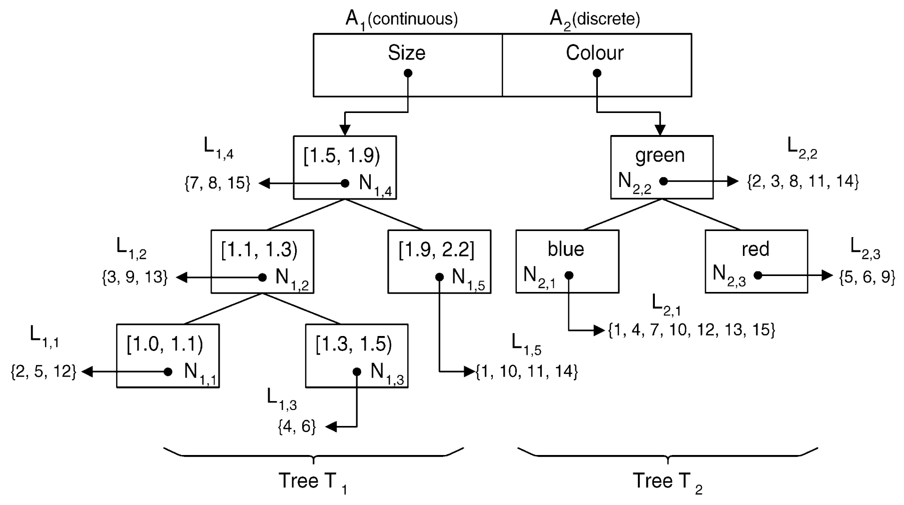


Fig. 4. Building the EES from a dataset with two attributes: continuous and discrete. EES is shown to the right. Each node in one tree has a list of indices to the examples of the dataset.

in the database. The rule-learning methods used by EAs invest approximately 85% of the time in evaluating the individuals (the average of a tenfold cross-validation with 20 UCI Repository databases [1]). This could be improved in two ways: designing a data structure that does not need to check every example in the dataset for each evaluation; and building models to estimate efficiently the individual fitness evaluation. Our work address the first direction.

IV. Description of the EES

The EES distributes the information from the dataset in such a way that it is possible to carry out the search into the space by attribute instead of by example. The data structure must be able to store this information regardless the type of attribute (continuous or discrete). In the case of continuous attributes, it is suitable to use a method to transform them into discrete ones, which reduce the cardinality of the set of values that this type of attributes can handle. In certain cases, the methods of rule generation apply discretization as a pre-processing of data in order to calculate such intervals. Given that an example should only belong to a single interval, it is required that the discretization method generates disjoint intervals.

The EES only contains information about several values for each attribute. The most interesting values are those that define the geometrical boundaries among labels. The example in Fig. 4 can clarify this. If the examples are ordered by the attribute *size*, we can observe that it is necessary to investigate only the values 1.1, 1.3, 1.5, and 1.9, because these are values which produce a change of class. Therefore, this coding allows to use all the possible intervals defined by every pair of cutpoints obtained by means of discretization, together with the range limits.

The EES would have a node with the interval [1.0, 1.1) for the attribute size: 1.1 is the mean value between 1.0 and 1.2 (as there is a change of label). This information is very useful to guide the genetic operators, specifically the mutation, since the mutation for real-valued attributes might generate any value in the range [1.0, 2.2]. Now, the number of possible mutations is limited by the boundary limits, (i.e., we only have 15 different intervals and, what it is more important: the list of examples that belongs to that interval can be associated to each node). This knowledge, incorporated into the EES will help the fitness function to evaluate individuals.

In general, for every attribute A_i in the dataset we will denote the finite set of values that A_i can take by Ω_i. In the case of A_i being a discrete attribute, Ω_i will contain values which will be represented as $V_{ij}(1 \le j \le |\Omega_i|)$. On the other hand, if we are dealing with continuous attributes, Ω_i will contain intervals which will be named $I_{ij}(1 \le j \le |\Omega_i|)$, the lower and upper bounds of which will be denoted by l_{ij} and u_{ij}, respectively.

The EES arranges the information from the dataset in a vector of binary and balanced search trees in such a way that the ith element of the vector will contain information about the ith attribute (A_i) in the dataset. Specifically, the different values or intervals that A_i can take are stored in one tree, which will be denoted by T_i. In addition to V_{ij} or I_{ij}, each node N_{ij} of the tree T_i contains a list (L_{ij}) of numbers which indicate the positions of examples in the database. If A_i is discrete, those indices contained in the list L_{ij} will correspond to those examples whose ith attribute take the jth value (V_{ij}) within the Ω_i set of possible values. If A_i is continuous, the indices contained in L_{ij} will correspond to those examples whose values for the ith attribute are included in the jth interval (I_{ij}) within the Ω_i set of possible intervals of such attribute. Fig. 4 shows an example of the data structure for a dataset with 15 examples and two attributes, where the first attribute is continuous, whereas the second is discrete. It is important to note that in case of continuous attributes the tree is sorted (in-order) by (disjoint) intervals and in case of discrete attributes it is alphabetically sorted by the discrete value. In this manner, any search within the tree has a logarithmic cost.

The fundamental property of the ESS is the possibility of accessing the information from the dataset through attributes instead of through examples. The main idea is to not have to process those examples whose values are not covered by the rule which is being evaluated. If we take a node N_{ij} of each T_i, the intersection of the lists L_{ij} will be the set of indices to examples for which each attribute A_i takes values that are covered by each node N_{ij}. The advantage that the EES offers lies in the fact that the intersections are carried out in an incremental manner, i.e., first the intersection of the list for the attribute A_1 and the list for the attribute A_2 is obtained. If such intersection is not empty, the list for the attribute A_3 is searched for and a new intersection between this and the result of the previous intersection is calculated. This process is repeated until all the attributes are completed, or until one of the intersections becomes empty. If the process concludes, the resulting list will contain the indices of the examples which fulfill the values or intervals of all the selected nodes N_{ij}.

The pseudocode of the evaluation algorithm using the EES is illustrated in Fig. 5. The function *EVALUATE* would be called every time an individual is going to be evaluated. Therefore, if the cost is lower than

```
Function EVALUATE
  Inputs:  R: Decision Rule; EES: Data Structure
  Output:  L: List of indices (examples covered by $R$)
 begin
   $i := 1$; $L_i := ListUnion(R, EES, i)$
   while $i < NumberOfAttributes(EES) \wedge L_i \neq \emptyset$
     $i := i + 1$
     $L_i := L_{i-1} \bigcap ListUnion(R, EES, i)$
   end while
   $L := L_i$
 end EVALUATE
 Function ListUnion
  Inputs:  R: Decision Rule; EES:Data Structure
           k: integer (attribute location)
  Output: L: List of indices (examples covered by $R_k$)
  begin
    $L := \emptyset$; $T_k := EES[k]$; ($T_k$: tree associated to attribute $A_k$)
    if $A_k$ is continuous
      for every $I_{kj} \in T_k \mid I_{kj} \subseteq R_k$
        $L := L \bigcup L_{kj}$
      end for
    else /*$A_k$ is Discrete */
      for every $V_{kj} \in T_k \mid V_{kj} \in R_k$
        $L := L \bigcup L_{kj}$
      end for
    end if
 end ListUnion
```

Fig. 5. Algorithm to evaluate individuals from the population by using EES.

NM (the cost of the linear vector), where N is the number of examples and M the number of attributes, this structure will be beneficial.

The notation used in the algorithm of Fig. 5 is the same as that used in the general structure of Fig. 4, excluding the following symbols: EES represents the data structure and R is the rule to be evaluated, while R_i is the condition that R establishes for the attribute A_i. The symbol L_i (with a single subindex) represents the list of accumulated intersections until the ith iteration, i.e., until the ith attribute. The function *EVALUATE* provides a single output parameter (L), which is the list of indices of examples resulting from the evaluation of the rule R over the data structure EES. In this way, the function *EVALUATE* calculates the intersections of the lists and returns the final list (L).

The main function uses an auxiliary function named *ListUnion*, that has the rule R, the structure EES and an integer that indicates the attribute to be evaluated. The only output parameter of this function is the list of indices L corresponding to the union of the lists L_{kj} for the attribute A_k (with $1 \leq j \leq |\Omega_k|$).

V. Experiments

The experiments carried out to test the efficiency of the data structure and the quality of decision rules consist of 15 different datasets from the UCI Repository. First, experiments are designed to show that EES is faster than the linear vector for any supervised learning system. Second, the evolutionary algorithm is run to obtain decision rules, and these are compared to that generated by the same evolutionary algorithm that uses the linear vector. Both groups of experimental results are summarized in Tables I and II, respectively.

For each dataset, groups of rules are randomly generated, although using a method that assures the uniform distribution of these rules. Subsequently, such rules are evaluated using the linear method and the data structure EES and the results obtained are compared. The linear evaluation method used in the tests is the most efficient one possible. For each rule, this method searches through the dataset, which has previously been stored in a vector of examples, processing each one of the examples. Likewise, for each example, the verification that its attributes fulfill the conditions of the rule is also carried out in a linear manner. However, it is not always necessary to process all the attributes of every example. If during the processing of an example, one of its values does not fulfill the condition that the rule establishes for the corresponding attribute, that example is no longer processed as it will no longer be able to fulfill the rule, regardless the values that the rest of the attributes takes, and therefore the next example is then processed.

TABLE I
Comparison of Average Results (Time in Seconds). The Average Evaluation Times of ESS and the Linear Vector Are Compared and the Improvement Is Shown in the Last Column

BD	A.T. ESS	A.T. Vector	Improv.(%)
breast cancer (Wisc.)	5.572	13.421	58.5
bupa liver disorder	1.464	4.370	66.5
cars	2.173	4.870	55.4
cleveland	3.460	6.042	42.7
glass	1.634	2.884	43.3
hayes–roth	0.720	1.499	52.0
heart desease	3.018	6.228	51.5
iris	0.517	1.536	66.3
led7	20.689	34.905	40.7
letter	222.556	842.784	73.6
pima indian	4.283	11.309	62.1
soybean	1.661	2.808	40.8
tic–tac–toe	7.623	11.031	30.9
vehicle	8.937	18.785	52.4
wine	2.084	4.120	49.4

Given that the major advantage of the structure lies in the evaluation of rules which do not cover examples, in order to ensure that the test method is fair, two types of rules have been generated and evaluated: valid and invalid rules (see the results varying the percentage of valid rules from 0% to 100% in Fig. 6). Let us call a rule valid if it covers at least one example from the dataset. In contrast, a rule which does not cover any of the examples in the dataset is called invalid. According to these definitions a valid rule will ensure that the structure is completely scanned from left to right, whereas in case of invalid rules the evaluation process of the data structure will be halted before the search has been completed. Although *a priori*, the evaluation of invalid rules seems to be pointless, this is not so, since the learning methods based on EAs generate intermediate rules which in many cases do not cover any example, mainly owing to the exploratory properties of genetic operators.

Table I shows the following: for each dataset (first column) the average time used by the EES (second column) and that used by the linear method (third column), as well as the improvement obtained by EES as compared with the vector-based evaluation (final column). For each dataset, the improvement is given by $(\mathrm{Time}(\mathrm{Vector}) - \mathrm{Time}(\mathrm{EES}) / \mathrm{Time}(\mathrm{Vector}))$, which represents the percentage of time saved by using the EES with respect to the time consumed by the vector in the average case. For all the datasets the average time taken by the EES is noticeably lower than that of the vector. The average improvement for the 15 datasets is 52.4%.

The experimental results are graphically shown in Fig. 6. This figure contains 15 graphs, one for each dataset used in the tests, which represent the evaluation time in seconds against the percentage of valid rules in the sets of rules evaluated. In turn, each graph contains two curves: the grey line shows the temporal results obtained for the evaluation using the vector of examples, while the black line refers to the results for the evaluation using the EES. Both representations show the temporal variation, while the percentage of valid rules is increased 10% step by step. For all the datasets, the behavior of the EES is very favorable in comparison to the vector of examples. On the other hand, as the percentage of valid rules increases, so does the rule evaluation time

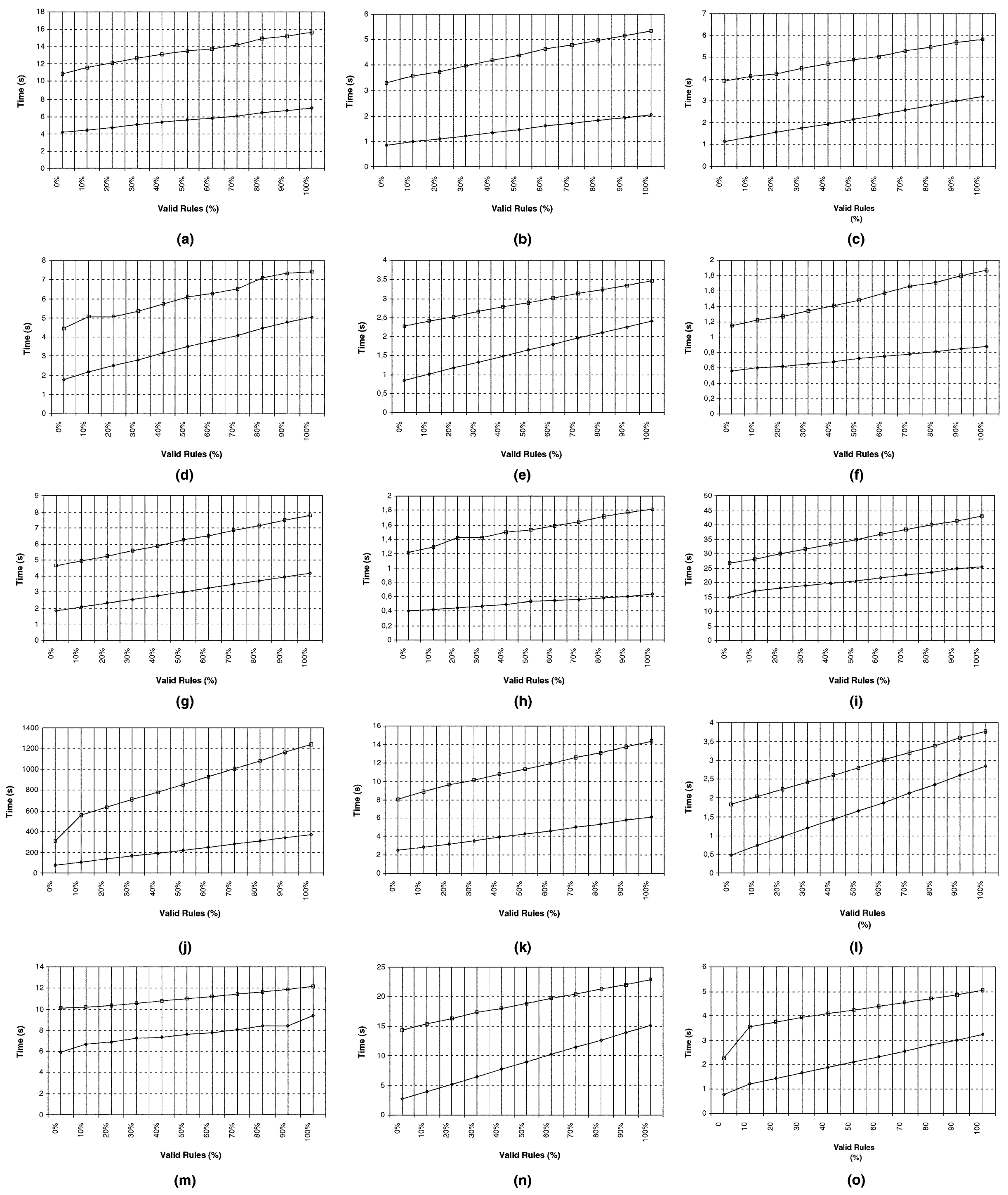


Fig. 6. Results for 15 datasets from UCI Repository. Light curve is the computational cost for the evaluation of the traditional vector of examples. Dark curve is the computational cost for EES. The x-axis is the percentage of valid rules used in the experiment and the y-axis is the time in seconds. (a) Breast Cancer. (b) Bupa Liver Disorder. (c) Cars. (d) Cleveland. (e) Glass. (f) Hayes-Roth. (g) Heart Diseases. (h) Iris. (i) Led7. (j) Letter. (k) Pima Diabetes. (l) Soybean. (m) Tic-Tac-Toe. (n) Vehicle. (o) Wine.

for both methods. The results show that the EES is highly efficient regardless the type of rule (valid or invalid), which gives an idea of the robustness of the structure.

Table II shows the performance of our approach. The same evolutionary algorithm has been evaluated by using the vector and the EES. The quality of the final sets of rules have been compared regarding the prediction accuracy and the number of rules. The averages are in the last row. The aim of this experiment is to experimentally prove that the EES improves the quality of the results. There is no statistical difference between the error rate provided by each method (17.8 and 17.9, respectively). However, the number of rules is decreased noticeably, by about half. This is mainly due to the effect of the EES on the genetic operators. The intervals included in the EES nodes drive the operators toward conditions that cover a greater number of examples without loss

TABLE II
COMPARISON OF AVERAGE RESULTS (LEARNING): ERROR RATE (ER) AND NUMBER OF RULES (NR)

	Linear Eval.		EES Eval.	
BD	ER	NR	ER	NR
breast cancer (Wisc.)	4.3	2.6	4.0	2.0
bupa	35.7	11.3	35.2	3.8
cars	14.3	16.2	11.9	8.1
cleveland	20.5	7.9	24.6	4.9
glass	29.4	19.0	33.8	9.8
hayes–roth	21.3	8.4	20.7	7.1
heart desease	22.3	9.2	21.8	4.3
iris	3.3	4.8	3.3	3.2
led7	27.9	41.9	27.0	42.3
letter	11.7	235.8	10.1	97.2
pima indian	25.9	16.6	25.6	5.1
soybean	11.7	47.2	1.8	4.0
tic–tac–toe	3.9	11.9	5.2	11.8
vehicle	30.6	36.2	34.2	16.9
wine	3.9	3.3	8.8	5.6

of accuracy. This fact allows us to classify the database with a fewer number of rules.

VI. CONCLUSION

The incorporation of knowledge into evolutionary algorithms is interesting for optimization and machine learning problems. Machine learning tasks, specially those related to supervised learning, need to process all the examples for the dataset a great number of times.

In this paper, we present a novel data structure (EES) with two goals: to incorporate knowledge to the EA to be used by genetic operators and to reduce the time complexity of the EA by means of a fast evaluation of examples from the dataset. EES incorporates knowledge in the early stage, when the structure is built before runing the EA, by organizing the discrete attributes and by obtaining potential good intervals for continuous attributes.

The main feature of the ESS is the possibility of accessing the information from the dataset through attributes instead of through examples. This means that EES organizes the information in such a way that it is not necessary to process all the examples to evaluate individuals (candidate decision rules) from the genetic population generated by a supervised learning system.

Results show that EES achieves to reduce the computational cost about 50%, maintaining the quality of decision rules generated. Likewise, the quality was similar to that obtained by using the vector, but the number of rules generated when using EES was about 50% less. Therefore, the approach guarantees the accuracy of the solutions taking less computational resources.

ACKNOWLEDGMENT

The authors are grateful to the reviewers for helpful comments and constructive criticisms.

REFERENCES

[1] J. S. Aguilar-Ruiz, J. C. Riquelme, and M. Toro, "Evolutionary learning of hierarchical decision rules," *IEEE Trans. Syst., Man, Cybern. B, Cybern.*, vol. 33, no. 2, pp. 324–331, Apr. 2003.

[2] J. S. Aguilar-Ruiz, D. Mateos, D. S. Rodriguez, and M. Ortiz, "Evolutionary neuroestimation of fitness functions," in *Proc. 11th Portuguese Conf. Artificial Intelligence (EPIA'03)*, Beja, Portugal, Dec. 2003.

[3] M. Annunziato and S. Pizzuti, "Adaptive parameterization of evolutionary algorithms driven by reproduction and competition," in *Proc. ESIT'2000*, Aachen, Germany.

[4] J. Bacardit and J. M. Garrell, "Evolving multiple discretizations with adaptive intervals for a Pittsburgh rule-based learning classifier system," in *Genetic and Evolutionary Computation Conf. 2003 (GECCO'03)*, Chicago, IL.

[5] J. L. Bentley and J. H. Friedman, "Data structures for range searching," *ACM Comput. Surv.*, vol. 11, no. 4, pp. 397–409, 1979.

[6] C. A. Coello and R. Landa, "Adding knowledge and efficient data structures to evolutionary programming: A cultural algorithm for constrained optimization," in *Proc. Genetic and Evolutionary Computation Conf. 2002 (GECCO'02)*, New York, Jul. 2002, pp. 201–209.

[7] K. A. DeJong, W. M. Spears, and D. F. Gordon, "Using genetic algorithms for concept learning," *Mach. Learn.*, vol. 1, no. 13, pp. 161–188, 1993.

[8] M. Freeston, "A general solution of the n-dimensional b-tree problem," in *Proc. 1995 ACM SIGMOD Int. Conf. Management of Data*, M. J. Carey and D. A. Schneider, Eds., May 1995, pp. 80–91.

[9] V. Gaede and O. Günther, "Multidimensional access methods," *ACM Comput. Surv.*, vol. 30, no. 2, pp. 170–231, 1998.

[10] R. Giráldez, J. S. Aguilar-Ruiz, J. C. Riquelme, J. F. Ferrer, and D. S. Rodríguez, "Discretization oriented to decision rule generation," in *Proc. Int. Conf. Knowledge-Based Intelligent Information and Engineering Systems (KES'02)*. Amsterdam, The Netherlands, 2002, pp. 275–279.

[11] R. Giráldez, J. S. Aguilar-Ruiz, and J. C. Riquelme, "Natural coding: A more efficient representation for evolutionary learning," in *Genetic and Evolutionary Computation Conf. 2003 (GECCO'03)*, Chicago, IL, Jul. 2003.

[12] D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*. Reading, MA: Addison-Wesley, 1989.

[13] A. Guttman, "R-trees: A dynamic index structure for spatial searching," in *Proc. ACM SIGMOD Int. Conf. Management of Data*, 1984, pp. 47–57.

[14] A. Henrich, H.-W. Six, and P. Widmayer, "The LSD tree: Spatial access to multidimensional point and nonpoint objects," in *Proc. 15th Int. Conf. Very Large Data Bases*, P. M. G. Apers and G. Wiederhold, Eds, Amsterdam, The Netherlands: Morgan Kaufmann, Aug. 1989, pp. 45–53.

[15] R. C. Holte, "Very simple classification rules perform well on most commonly used datasets," *Mach. Learn.*, vol. 11, pp. 63–91, 1993.

[16] H. V. Jagadish, "Spatial search with polyhedra," in *Proc. 6th Int. Conf. Data Engineering*, Los Angeles, CA, Feb. 5–9, 1990, pp. 311–319.

[17] C. Z. Janikow, "A knowledge-intensive genetic algorithm for supervised learning," *Mach. Learn.*, vol. 1, no. 13, pp. 169–228, 1993.

[18] Y. Jin, *Knowledge in Evolutionary and Learning Systems*. Aachen, Germany: Shaker Verlag, 2002.

[19] Y. Jin, M. Olhofer, and B. Sendhoff, "A framework for evolutionary optimization with approximate fitness functions," *IEEE Trans. Evol. Comput.*, vol. 6, no. 5, pp. 481–494, Oct. 2002.

[20] Y. Jin, "A comprehensive survey of fitness approximation in evolutionary computation," *Soft Comput. J.*, vol. 9, no. 1, pp. 3–12, 2005.

[21] S. J. Louis and G. J. E. Rawlins, "Designer genetic algorithms: Genetic algorithms in structure design," in *Proc. 4th Int. Conf. Genetic Algorithms*, R. Belew and L. Booker, Eds. San Mateo, CA: Morgan Kauffman, 1991, pp. 53–60.

[22] S. J. Louis, "Genetic Algorithms as a Computational Tool for Design," Ph.D. dissertation, Dept. Comput. Sci., Indiana Univ., Bloomington, Aug. 1993.

[23] S. J. Louis and F. Zhao, "Domain knowledge for genetic algorithms," *Int. J. Expert Syst.*, vol. 8, no. 3, pp. 195–211, 1995.

[24] Z. Michalewicz and C. Z. Janikow, "Handling constraints in genetic algorithms," in *Proc. 4th Int. Conf. Genetic Algorithms (ICGA'91)*, R. K. Belew and L. B. Booker, Eds., San Diego, CA, 1991, pp. 151–157.

[25] A. W. Moore and M. S. Lee, "Cached sufficient statistics for efficient machine learning with large datasets," *J. Artif. Intell. Res.*, vol. 8, pp. 67–91, 1998.

[26] J. Nievergelt, H. Hinterberger, and K. C. Sevcik, "The grid file: An adaptable symmetric multikey file structure," *ACM Trans. Database Syst., ACM CR 8411-0931*, vol. 9, no. 1, 1984.

[27] B. C. Ooi, "Spatial KD-tree: A data structure for geographic database," in *BWT*, 1987, pp. 247–258.

[28] D. J. Powell, S. S. Tong, and M. M. Skolnick, "EnGENEous domain independent, machine learning for design optimization," in *Proc. 3rd Int. Conf. Genetic Algorithms*, J. D. Schaffer, Ed. San Mateo, CA: Morgan Kaufmann, 1989, pp. 151–159.

[29] *Handbook of Genetic Algorithms*, L. Davis, Ed., Van-Nostrand Reinhold, New York, 1991, pp. 312–331. D. J. Powell, M. M. Skolnick, S. S. Tong. Interdigitation: A Hybrid Technique for Engineering Design Optimization Employing Genetic Algorithms, Expert Systems, and Numerical Optimization.

[30] R. Quinlan. (1998–2001). [Online]. Available: http://www.rulequest.com

[31] G. J. E. Rawlins, *Foundations of Genetic Algorithms*. San Mateo, CA: Morgan Kaufmann, 1991.

[32] R. G. Reynolds, "An introduction to cultural algorithms," in *Proc. 3rd Annu. Conf. Evolutionary Programming*, A. V. Sebald and L. J. Fogel, Eds. River Edge, NJ: World Scientific, 1994, pp. 131–139.

[33] ——, "Cultural algorithms: Theory and applications," in *New Ideas in Optimization*, D. Corne, M. Dorigo, and F. Glover, Eds, London, U.K.: McGraw-Hill, 1999, pp. 367–377.

[34] J. T. Robinson, "The K-D-B-Tree: A search structure for large multidimensional dynamic indexes," in *Proc. 1981 ACM SIGMOD Int. Conf. Management of Data*, Y. E. Lien, Ed. Ann Arbor, MI: ACM Press, 1981, pp. 10–18.

[35] S. Ruggieri, "Efficient C4.5," *IEEE Trans. Knowl. Data Eng.*, vol. 14, no. 2, pp. 438–444, Apr. 2002.

[36] S. B. Hamida and M. Schoenauer, "An adaptive algorithm for constrained optimization problems," in *Proc. Parallel Problem Solving from Nature VI*. Berlin, Germany: Springer-Verlag, 2000, pp. 529–538. LNCS 1917.

[37] K. Shim, *SIGKDD Explorations*, vol. 2, no. 2, Dec. 2000.

[38] R. Thomsen, G. B. Fogel, and T. Krink, "A clustal alignment improver using evolutionary algorithms," *Proc. 4th Congr. Evolutionary Computation (CEC'2002)*, vol. 1, pp. 121–126, 2002.

[39] G. Venturini, "SIA: A supervised inductive algorithm with genetic search for learning attributes based concepts," in *Proc. European Conf. Machine Learning*, 1993, pp. 281–296.

Multiobjective GA Optimization Using Reduced Models

Deepti Chafekar, Liang Shi, Khaled Rasheed, and Jiang Xuan

***Abstract*—In this paper, we propose a novel method for solving multiobjective optimization problems using reduced models. Our method, called objective exchange genetic algorithm for design optimization (OEGADO), is intended for solving real-world application problems. For such problems, the number of objective evaluations performed is a critical factor as a single objective evaluation can be quite expensive. The aim of our research is to reduce the number of objective evaluations needed to find a well- distributed sampling of the Pareto-optimal region by applying reduced models to steady-state multiobjective GAs. OEGADO runs several GAs concurrently with each GA optimizing one objective and forming a reduced model of its objective. At regular intervals, each GA exchanges its reduced model with the others. The GAs use these reduced models to bias their search toward compromise solutions. Empirical results in several engineering and benchmark domains comparing OEGADO with two state-of-the-art multiobjective evolutionary algorithms show that OEGADO outperformed them for difficult problems.**

***Index Terms*—Genetic algorithms, multiobjective optimization, reduced models.**

I. Introduction

This paper concerns the application of reduced models for constrained multiobjective genetic algorithm (GA) optimization. The GA presented in this paper is mainly aimed at solving problems from realistic engineering design domains that usually involve simultaneous optimization of multiple and conflicting objectives with many constraints. In these problems instead of a single optimum there is usually a set of trade-off solutions called the nondominated solutions or Pareto-optimal solutions (also called Pareto front). For such solutions no improvement in any objective is possible without sacrificing at least one of the other objectives. No other solutions in the search space are superior to these Pareto-optimal solutions when all objectives are considered.

Many challenges are faced in the application of GAs to engineering design domains. A large number of objective evaluations may be required in order to obtain trade-off Pareto-optimal solutions. Moreover, the search space can be complex with many constraints and a small feasible (physically realizable) region. However, determining the quality (fitness) of each point may involve the use of a simulator or an analysis code that takes a long time. Therefore, it is impossible to be cavalier with the number of objective evaluations in an optimization.

For such problems, multiobjective evolutionary algorithms are preferable as they can find the Pareto front in one run. Many evolutionary algorithms for solving multiobjective optimization problems have been developed. The most recent ones are the ε-multiobjective evolutionary algorithm (ε-MOEA) [5], nondominated sorting genetic algorithm-II (NSGA-II) [3], strength Pareto evolutionary algorithm-II (SPEA-II) [16], and Pareto envelope-based selection-II (PESA-II) [2]. Most of these approaches propose the use of a generational GA. The ε-MOEA proposed by Deb is a steady-state MOEA based on the ε-dominance concept. The main aim of these methods is obtaining a well-converged and well-distributed Pareto front. There usually exists a tradeoff in these methods between obtaining a well-distributed Pareto front and the number of objective evaluations performed. Many real-world application problems are computationally complex and performing a large number of objective evaluations on these problems may be very difficult. The ε-MOEA method proposed by Deb is a fast multiobjective evolutionary algorithm in terms of computational time. The goal of our research, however, is the development of a method that: 1) converges close to the true Pareto front; 2) finds a well-distributed Pareto front; and 3) performs fewer objective evaluations.

In this paper, we propose a novel method for multiobjective optimization based on the use of a steady-state GA and reduced models. This method is relatively fast and practical. It is also easy to transform a single-objective GA to a multiobjective GA by using our method.

Our method can be viewed as a multiobjective transformation of the genetic algorithm for design optimization (GADO) [9], [12], a GA that was designed with the goal of being suitable for use in engineering design. It uses new operators and search control strategies that target engineering domains [12]. GADO has been successfully applied to a variety of optimization tasks, which span many fields. It demonstrated a great deal of robustness and efficiency relative to competing methods [9].

In GADO, each individual in the GA population represents a parametric description of an artifact. The fitness of each individual is based on the sum of a proper measure of merit computed by a simulator or some analysis code, and a penalty function if relevant. A steady-state model is used, in which several crossover and mutation operators are applied to two parents selected by linear rank based selection. One offspring point is produced, and then an existing point in the population is replaced by the newly generated point. The replacement strategy is a crowding technique, which takes into consideration both the fitness and the proximity of the points in the GA population. GADO monitors the degree of diversity of the GA population. If, at any stage, it is

Manuscript received September 15, 2003; revised February 3, 2004 and April 13, 2004. This work was supported by the National Science Foundation under Grant CTS-0121058. This paper was recommended by Guest Editor Y. Jin.

D. Chafekar is with the Computer Science Department, Virginia Polytechnic Institute and State University, Blacksburg, VA 24061 USA (e-mail: chafekar@vt.edu).

L. Shi and K. Rasheed are with the Computer Science Department, University of Georgia, Athens, GA 30602 USA (e-mail: shi@cs.uga.edu; khaled@cs.uga.edu).

J. Xuan is with the Computer Science Department, Stanford University, Stanford, CA 94305 USA (e-mail: jakexuan@hotmail.com).

Digital Object Identifier 10.1109/TSMCC.2004.841905