Natural Evolutionary Coding: an application to estimating Software Development Projects

Jesús S. Aguilar–Ruiz
Department of Computer Science
University of Seville
aguilar@lsi.us.es

José C. Riquelme
Department of Computer Science
University of Seville
riquelme@lsi.us.es

Isabel Ramos
Department of Computer Science
University of Seville
isabel.ramos@lsi.us.es

Abstract

Software Project Simulator and Evolutionary Computation are combined to generate decision rules. The purpose is to provide accurate decision rules in order to help the project manager to take decisions at any time in the development. To obtain these rules we have analysed a new method to encode the individuals (rules) of the genetic population. Our approach, named "natural coding", uses one gene per continuous attribute leading to a reduction in the search space, what might have influence on the convergence of the evolutionary algorithm. Genetic operators for this approached natural coding are described. The application of our method to databases obtained by means of simulations are summarized in the last section, in which the decision rules are compared to that of C4.5. Results show that the evolutionary approach produces better decision rules for this decision–making task.

1 INTRODUCTION

In problems related to supervised learning, the decision rules are especially relevant. Given a database with continuous attributes, and a class label, we try to find a rule set that describes the knowledge within data or classify new unseen data. For an attribute a_i, the rules take the form of "if $a_j \in [l_j, u_j]$ then *class*", where l_j and u_j are two real values belonging to the range of the attribute and $l_j < u_j$. For example, we assume that we have an hypothetical database that associate the weight (in kilograms) and height (in meters) of a person with being or not candidate to have accepted a paper in a relevant conference. The database is a sequence of tuples such as (83, 1.71, no), (71, 1.62, yes), etc. A rule describing the relationship among attribute values and class might be: *if weight* $\in$ *[60, 70] height* $\in$ *[1.60,1.68] then he/she is a candidate.*

The finding of these rules can be tackled with many different techniques and the evolutionary algorithms are among them. Two critical factors have influence on the decision rules obtained by an evolutionary algorithm: the selection of an internal representation of the search space (*coding*) and the definition of an external function that assigns a value of goodness to the potential solutions (*evaluation*).

In this work, we are especially interested in the coding, i.e. in finding a method to accurately encode the genetic information of the individuals. The coding method, named "natural coding" because it uses natural numbers, needs only one value (gene) per attribute. Every interval is encoded by one natural number. This coding needs a new definition for the genetic operators in order to avoid the convertion from natural numbers to the values of the original space (from genotype to phenotype). These definitions are presented and it is shown in the paper, the evolutionary algorithm can work directly with the natural coding until the end, when the individuals will decoded to decision rules. Our approach leads to a reduction of the search space size, what has a positive influence on the convergence of the evolutionary algorithm.

This natural coding has been applied to the management of Software Development Projects and the results compared to that of C4.5 (Quinlan, 1993) are very satisfactory. C4.5 is a well–known tool which basically consists in a recursive algorithm with divide and conquer technique that optimises the tree construction on basis to gain information criterion. The program output is a graphic representation of the found tree, a confusion matrix from classification results and an estimated error rate. As for number of rules as number of records covered by the rules the evolutionary algorithm found decision rules with higher quality.

The dynamic models for software projects have a set of initial parameters that define the management policies to be applied. These policies are associated with the organization (maturity level, average delay, turnover of the project's work force, etc.) and the project (number of tasks, time, cost, number of technicians, software complexity, etc.). The use of an Software Project Simulators (SPS) will be complex if the number of parameters is very large. For example, the dynamic model shown in (Abdel–Hamid, 91) has about 60 parameters. Therefore, the impact on the project of 60 different features could be known. The more parameters the model has, the more complex the development of the project and the use of the model.

Decisions made in any organizational setting are based on what information is actually available to the decision–makers. The computer simulation tools of dynamical systems provide us with the possibility of changing one or several factors while the remaining ones are kept unchanged. The implications of managerial policies on the software development process could be analysed to infer the best decision. The SPS plays an important role in the decision–making task: first, post-mortem projects can be analysed in order to infer which actions could improve the results; second, an a priori analysis would indicate the intervals within which the values of the parameters have a tendency to achieve the aims of the project.

Other modelling techniques have been applied for estimating effort or cost of software projects: ordinary least-squares regression, analogy-based estimation (Finnie el al., 2000), (Shepperd an Schofield, 2000), (Walkerden and Jeffery, 1999), genetic programming (Dolado, 2001). Nevertheless, evolutionary algorithms provide us with a method for finding good solutions (in our case, decision rules) in a complex search space (database generated by an SPS) where parameters do not have an obvious relationship (Aguilar et al., 2001) . In Section 2 the new coding method for evolutionary algorithm is presented, together with the specific definition of genetic operators. Section 3 describes briefly the more relevant aspect of the evolutionary algorithm. The experiments were carried out by defining three different artificial scenarios and they are analysed in Section 4.

2 NATURAL CODING

Through the text, we will use a very simple database in order to explain the application of the genetic operators. This database has one attribute with range [1.1,6.2].

Nevertheless, the real coding is more appropriate with real domains, simply because is more natural to the domain. A number of authors have investigated non–binary evolutionary algorithms theoretically (Bhattacharyya and Koehler, 1994), (Koehler et al., 1998), (Vose et al., 1998) . In this sense, each gene would be encoded with a float value. Two float values would be needed to express the interval of a continuous attribute.

The number of values that the attribute can take in its real range is infinite. The search space is therefore large, so that reducing this search space would make possible a faster convergence of the algorithm . Then, the first step consists in trying to diminish the cardinality of the set of values of the attribute.

2.1 Diminishing the cardinality

A number of remarkable supervised discretization methods has been approached in the bibliography. Among them, the Holte's 1R (Holte, 1993) and the method of Fayyad and Irani (Fayyad and Irani, 1993) are well–known. However, as the aim of this method is not to find intervals but cut–points to be used as limits of the further decision rules, we assume that any supervised discretization method would be appropriate for our purpose. As we will see below, if the discretization method produces k intervals, then there will be $k+1$ cut–points and will therefore be $\binom{k+1}{2}$ possible intervals for the decision rules.

Our goal consists in observing the class along the discretization method in order to decrease the alphabet size. This coding allows to use all the possible intervals defined by every two cutpoints obtained by means of discretization.

Firstly, we will analyse what intervals inside the range of the attribute tends to appear as intervals for a possible decision rule obtained from the natural coding. This task could be solve by any supervised discretization algorithm, for example, the well–known method $1R$ proposed in (Holte, 1993) . Once the vector indicating which are the boundaries for the intervals is obtained (*vector of cuts*), we assign natural numbers to any possible combination, as it appears in Table 1.

Example 1 *Let assume from the database example that the output of the discretization algorithm is the vector of cuts* $\{1.1, 3.75, 4.85, 5.2, 6.2\}$. *The possible intervals to be generated from those values are shown in Table 1. Each interval is identified by a natural*

Table 1: Intervals calculated for the continuous attribute with range $[1.1, 6.2]$.

Cut–points	3.75	4.85	5.2	6.2
1.1	**1** $\equiv [1.1, 3.75]$	**2** $\equiv [1.1, 4.85]$	**3** $\equiv [1.1, 5.2]$	**4** $\equiv [1.1, 6.2]$
3.75	-	**6** $\equiv [3.75, 4.85]$	**7** $\equiv [3.75, 5.2]$	**8** $\equiv [3.75, 6.2]$
4.85	-	-	**11** $\equiv [4.85, 5.2]$	**12** $\equiv [4.85, 6.2]$
5.2	-	-	-	**16** $\equiv [5.2, 6.2]$

number, for example, the interval $[3.75, 4.85]$ *will be referenced by the natural number 6.*

Table 1 shows 5 cutpoints, which can generate 10 intervals. The number of intervals defines the size of the alphabet for such attribute and depends on the number of cuts k, exactly $|\Omega| = \frac{\mathsf{k}\,(\mathsf{k}-1)}{2}$. To know the maximum number of cuts that an attribute should have in order to consider interesting this coding is $\mathsf{k} < \sqrt{3n}$, where n is the number of different values.

In Table 1 a natural number (in bold), beginning by 1, from left to right and from top to bottom, is assigned to each interval. These “natural” numbers will help us to identify such intervals later.

2.2 Transitions

Once the necessary number of cuts has been calculated, we know the size of the new alphabet $|\Omega|$. From now, we will analyse the mutation and crossover operators for this coding.

To know what interval could be obtained from another, the figures from Table 1 are placed into a graph (see Figure 1). Every state is indicating graphically what are their adyacent states. The number of states will be the size of the new alphabet. All these values taken from a given state are not erroneous, i.e. every interval obtained from a mutation or crossover will be coherent. To assure that each interval is coherent for all the generations is a good step to achieve a fast convergence.

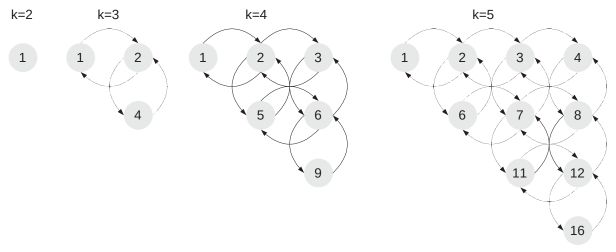

Figure 1: Transitions for $\mathsf{k} = 2$, $\mathsf{k} = 3$, $\mathsf{k} = 4$ and $\mathsf{k} = 5$.

Definition 1 (row and column) *Let n be the value of the gene, and let r and c be the row and the column, respectively, where n is located in Table 1. The way in which r and c are calculated is: (% is the remainder of the integer division)*

$$r = \frac{n-1}{\mathsf{k}-1} + 1 \qquad c = (n-1)\%(\mathsf{k}-1) + 1 \tag{1}$$

Example 2 *Let* $n_i = 2$ *and* $n_j = 8$. *Then*

$$r(2) = 1 \quad c(2) = 2 \quad r(8) = 2 \quad c(8) = 4$$

That is to say, 2 is in row 1 and column 2, and 8 is in row 2 and column 4.

Definition 2 (boundaries) *Let n be the value of the gene, we named boundaries of the value n to those values from Table 1 that limits the four possible shifts (one by direction): left, right, up and down, and they will be denoted as ble, bri, bup and bdo, respectively, and they will be calculated as:*

$$\begin{array}{l} ble(n) = (\mathsf{k}-1)(r-1) + r \\ bri(n) = (\mathsf{k}-1)r \\ bup(n) = c \\ bdo(n) = (\mathsf{k}-1)(c-1) + c \end{array} \tag{2}$$

Example 3 *From Example 2,*

$$\begin{array}{llll} ble(2) = 1 & bri(2) = 4 & bup(2) = 2 & bdo(2) = 6 \\ ble(8) = 6 & bri(8) = 8 & bup(8) = 4 & bdo(8) = 16 \end{array}$$

That is to say, 2 could reach up to 1 to the left, up to 4 to the right, up to 2 to the top and up to 6 to the bottom; 8 could reach up to 6 to the left, up to 8 to the right, up to 4 to the top and up to 16 to the bottom.

Definition 3 (shifts) *The left, right, up and down adjacent shifts for a value n will be obtained (if possible) as follows:*

$$\begin{array}{l} left(n) = max(ble(n), n-1) \\ right(n) = min(bri(n), n+1) \\ up(n) = max(bup(n), n-\mathsf{k}+1) \\ down(n) = min(bdo(n), n+\mathsf{k}-1) \end{array} \tag{3}$$

We define horizontal and vertical shifts as all the possible shifts as for row as for column, respectively, where

n is placed, including n.

$$\overline{hor}(n) = \bigcup_{i=1}^{\mathsf{k}-1} max(ble(n), (\mathsf{k}-1)(r-1)+i) \quad (4)$$

$$\overline{ver}(n) = \bigcup_{i=1}^{\mathsf{k}-1} min(bdo(n), (\mathsf{k}-1)(i-1)+c) \quad (5)$$

Example 4 *From Example 3, the adjacent shifts of 2 and 8 will be:*

left(2)=1	*right(2)=3*	*up(2)=2*	*down(2)=6*
left(8)=7	*right(8)=8*	*up(8)=4*	*down(8)=12*

$\overline{hor}(2) = \{1,2,3,4\}$ $\quad \overline{ver}(2) = \{2,6\}$
$\overline{hor}(8) = \{6,7,8\}$ $\quad \overline{ver}(8) = \{4,8,12,16\}$

2.3 Natural mutation

A mutation consists in selecting a near interval to that that has the value n. For example, observing Table 1, if the number of cuts is equal to 5 ($\mathsf{k}=5$), and $n=7$, there are four possible mutations {3,6,8,11}; however, if $n=4$, there will be two possible mutations {3,8}.

Definition 4 (natural mutation) *Let n be the value of the gene, the natural mutation of n, denoted by $Mut(n)$, is any near value to n by using the shifts and distinct from n.*

$$Mut(n) \in \{x \mid x \in \{mov(n) - n\}\} \quad (6)$$

where $mov(n) = left(n) \cup right(n) \cup up(n) \cup down(n)$.

Example 5 *Thus, $Mut(2) \in \{\{1,2,3,6\} - \{2\}\}$, i.e., $Mut(2) \in \{1,3,6\}$. Now, one of the three values is selected.*

2.3.1 Natural crossover

Definition 5 (natural crossover) *The natural crossover between two values n_i and n_j, denoted by $Cruce(n_i, n_j)$ is obtained as follows:*

$$Cross(n_i, n_j) \in$$

$$\in \left(\left(\overline{hor}(n_i) \cap \overline{ver}(n_j)\right) \cup \left(\overline{hor}(n_j) \cap \overline{ver}(n_i)\right)\right) \quad (7)$$

We can observe in Table 1 that the nearest values are in the intersection between the row and the column where both values being crossed are placed. For example, the nearest value to 1 and 6 is 2; the nearest value to 6 and 12 is 8. Only when the values n_i and n_j are located in the same row or column the interval will be inside the other.

Example 6 *Thus,*
$Cross(2,8) \in$
$\{\{\{1,2,3,4\} \cap \{4,8,12,16\}\} \cup \{\{6,7,8\} \cap \{2,6\}\}\}$,
i.e., $Cross(2,8) \in \{4,6\}$.
Now, we can select one or more of the values generated by the crossover operator.

3 ALGORITHM

The algorithm is a typical sequential covering EA (Mitchell, 1997) . It chooses the best individual of the evolutionary process, transforming it into a rule which is used to eliminate data from the training file (Venturini, 1993) . In this way, the training file is reduced for the following iteration. A termination criterion could be reached when more examples to cover do not exist. The method of generating the initial population consists in randomly selecting an example (with the label of interest) from the training file for each individual of the population. Afterwards, an interval to which the example belongs is obtained by adding and subtracting a small random quantity from the values of the example.

The fitness function must be able to discriminate between correct and incorrect classifications of examples. Finding an appropriate function is not a trivial task, due to the noisy nature of most databases.

The evolutionary algorithm maximizes the fitness function f for each individual. It is given by the equation 8.

$$f(i) = 2(N - CE(i)) + G(i) + coverage(i) \quad (8)$$

where N is the number of examples being processed; $CE(i)$ is the class error, which is produced when the example i belongs to the region defined by the rule but does not have the same class; $G(i)$ is the number of examples correctly classified by the rule; and the *coverage* of the i^{th} rule is the proportion of the search space covered by such rule. Each rule can be quickly expanded to find more examples thanks to the coverage in the fitness function.

4 EXPERIMENTS

By simulating the Abdel–Hamid's dynamical model (Abdel–Hamid, 1991) we have obtained three different scenarios. These scenarios are defined by the intervals of values that the attributes can take. Concretely, the initial values were almost the same as those described in the Abdel–Hamid's work. Only four attributes were modified, taking random values from previously defined intervals. Each row contains the name of the attribute in the model, the range of values it can assume, a brief description of its meaning and the estimated value at the beginning of the project.

The attributes ASIMDY, HIREDY, TRNSDY (related with the new personnel management enrolled to the ongoing project) and MXSCDX (related with the decision of imposing constraints to the delivery time) described in Tabla 2, will allow us to analyse their influence on the variables of the project (mainly, delivery time, cost or effort, and quality) described in Table 3. Table 2 show attribute names, description, range and initial estimated value for the attributes or input parameters. Table 3 show attribute names, description and initial estimated value for the variables or output parameters. These input and output parameters will generate the three case of study:

- Scenario 1: Attributes can take any value within the range (defined in Table 2).
- Scenario 2: Attributes related with the personnel management take the low values in their range, i.e. ASIMDY in [5,15], HIREDY in [5,10] and TRNSDY in [5,10], what means that the personnel management is fast. Moreover, the date extension can not be greater than 20% of the initially estimated value.
- Scenario 3: Personnel management is the same as before, and MXSCDX can take any value in its range.

The goal is to find which management policies produce good results for the variables of interest in every scenario. In this work, the quality must be prioritized, only considering those projects for which the variable QUALITY has a low value. From these constraints, we try to obtain policies that minimize the delivery time (independently of the effort), the effort (without taking into account the delivery time) or both delivery time and effort at once.

Decision rules for each scenario are obtained by the following steps:

- Define the intervals of values for the attributes of the dynamical model (see Table 2).
- Define the goals of the project (values for time and cost).
- Generate the database: Each simulation produces a record with the values of the parameters and the values of the variables shown in Table 3 and this record is saved in a file.
- Assign labels to the records according to a threshold for every variable.
- From the file generated in the preceding step, a set of decision rules is provided automatically by the evolutionary algorithm for the decision-making task.
- These decision rules are compared with those produced by C4.5.

Table 4 gives the values of the parameters involved in the evolutionary process.

Table 4: Parameters of the evolutionary algorithm.

Parameter	**Value**
Population size	100
Generations	50
Crossover probability	0.5
Individual Mutation probability	0.2
Gene mutation probability	0.1

It is worth noting that the decision rules presented below were obtained by running the evolutionary algorithm with 50 generations. This means that the running time of the algorithm was very small, less than a minute in a Pentium II 450MHz.

4.1 SCENARIO 1

The results for this scenario are as follows: the variables EFFORT take values in [1589,3241], TIME in [349.5,437] and QUALITY in [0.297,0.556]. In order to assign a label to the records, the variables were discretized, taking one threshold for TIME and EFFORT and two for QUALITY. The constraints over the attributes and variables are shown in Table 5, where out of three hundred simulations only 8 had quality less than 0.35, whereas when QUALITY is less than 0.45 there were 26 records with EFFORT under 1888 and 27 records with TIME under 384. Only one record satisfied the constraints for both EFFORT and TIME simultaneously.

We use the evolutionary algorithm to find rules that provide an adequate effort, a good delivery time or an excellent quality (three independent analysis).

- Rule for effort: a rule covering 22 records (C4.5 produces two rules covering 23 records).

  if 101$\leq$ ASIMDY and
  and 20.2$\leq$ HIREDY and
  and 6.9$\leq$ MXSCDX then
  then EFFORT$\leq$ 1888 and QUALITY$\leq$ 0.45

Table 2: Parameters of the environment of the project and organization.

Attribute	Interval	Description	Estimated Value
ASIMDY	[5,120]	Average assimilation delay (days)	20
HIREDY	[5,40]	Average hiring delay (days)	30
TRNSDY	[5-15]	Time delay to transfer people out (days)	10
MXSCDX	[1-50]	Maximum schedule completion date extension (dimimensionless)	3

Table 3: Variables of the environment of the project and organization.

Variable	Description	Estimated Value
EFFORT	Cost or necessary effort for the project development (tech. per day)	1.111
TIME	Delivery time (days)	320
QUALITY	Quality of the final product (errors/task)	0

Table 5: Number of Software Development Projects satisfying the constraints. Rows indicate constraints over variables and columns over attributes: (1) None; (2) EFFORT < 1888; (3) TIME < 384.

Constraint	(1)	(2)	(3)
NONE	300	42	37
QUALITY < 0.35	8	1	8
QUALITY < 0.45	227	26	27

If the assimilation and the hiring are slow then the effort is optimized.

- Rule for time: a rule covering 20 records (C4.5 produces two rules covering 21 records).

  if 7$\leq$ ASIMDY$\leq$ 17.9 and
  and 5.6$\leq$ HIREDY$\leq$ 39.5 and
  and 2.3$\leq$ MXSCDX and
  and 5.4$\leq$ TRNSDY then
  then TIME$\leq$ 384 and QUALITY$\leq$ 0.45

  These rule shows that the intervals for the attributes HIREDY, MXSCDX and TRNSDY are very unrestricted, i.e. these attributes might take any value within the range. Therefore, the rule could consider only the attribute ASIMDY: a fast assimilation is enough to fulfill the time constrains.

- Rule for quality: if QUALITY must be less than 0.35, only 8 records with TIME less than 384 are obtained. The evolutionary algorithm needs one rule (C4.5 needs two).

  if ASIMDY$\leq$ 23.2 and
  and 5.9$\leq$ HIREDY$\leq$ 32.8 and
  and 13.1$\leq$ MXSCDX$\leq$ 47.1 and
  and 5.5$\leq$ TRNSDY$\leq$ 10.8 then
  then TIME$\leq$ 384 and QUALITY$\leq$ 0.35

  This rule is similar as before but warning that the average delay to transfer people out must be lower.

Table 6: Number of Software Development Projects satisfying the constraints. (1) None; (2) EFFORT < 1999; (3) TIME < 352.

Constraint	(1)	(2)	(3)
NONE	300	105	297
QUALITY < 0.35	13	0	12
QUALITY < 0.45	47	1	45

4.2 SCENARIO 2

For this scenario the variable EFFORT takes values in [1709,3395], TIME in [349.5, 354.3] and QUALITY in [0.235, 0.661]. The thresholds for labelling the database were 1999 for EFFORT, 352 for TIME and 0.35 and 0.45 for QUALITY. The number of records satisfying these constraints are shown in table 6.

- Rules for time: two rules covering 31 records out of 45. (C4.5 produces 3 rules covering 22 records).

  if 11.6 $\leq$ ASIMDY and
  and 6.3 $\leq$ HIREDY and
  and 1.1$\leq$ MXSCDX $\leq$ 1.15 then
  then TIME$\leq$ 352 and QUALITY$\leq$ 0.45

  if 8.6 $\leq$ ASIMDY and
  and 6.8 $\leq$ HIREDY $\leq$ 9.5 and
  and 1.16 $\leq$ MXSCDX then
  then TIME $\leq$ 352 and QUALITY $\leq$ 0.45

  These rules are complementary with regard to the schedule completion date extension: the first one has a fixed schedule and the second one has not.

- Rules for quality: for 12 records with QUALITY less than or equal to 0.35 the evolutionary algorithm produced a rule covering six of them (C4.5 produced three rules covering 8 out of 12):

Table 7: Number of Software Development Projects satisfying the constraints. (1) None; (2) EFFORT < 1999; (3) TIME < 352.

Constraint	**(1)**	**(2)**	**(3)**
NONE	300	164	226
QUALITY < 0.35	43	0	9
QUALITY < 0.45	116	11	49

if $11.8 \leq$ ASIMDY and
and $7.8 \leq$ HIREDY ≤ 9.8 and
and $1.1 \leq$ MXSCDX ≤ 1.19 and
and TRNSDY ≤ 6.8 then
then TIME ≤ 352 and QUALITY ≤ 0.35

To improve the quality, the rule must limit the time delay to transfer people out.

4.3 SCENARIO 3

The model is again simulated 300 times with the following values for the variables: EFFORT in [1693,2415], TIME in [349.5,361.5] and QUALITY in [0.236,0.567]. The threshold for labelling the database are the same that in the second scenario. The number of records satisfying the constraints is shown in Table 7.

- Rules for effort and time: the scenario 3 is the only one that has a case in where both constraints over time and effort are sastisfied: When EFFORT<1999 and QUALITY<0.45 there are 11 cases matching TIME <352 by chance. Two rules cover these records (C4.5 needs three rules for 9 out of 11).

  if $8.4 \leq$ ASIMDY ≤ 11.5 and
  and $8.8 \leq$ HIREDY and
  and $9.3 \leq$ MXSCDX then
  then EFFORT ≤ 1999 and TIME ≤ 352 and
  and QUALITY ≤ 0.45 (7 records)

  if $9.8 \leq$ ASIMDY ≤ 11.2 and
  and $6.8 \leq$ HIREDY ≤ 8 and
  and MXSCDX ≤ 39.5 then
  then EFFORT ≤ 1999 and TIME ≤ 352 and
  and QUALITY ≤ 0.45 (4 records)

  These rules point out that for obtaining good results simultaneously for TIME and EFFORT is essential that the assimilation of technicians is fast. The two rules are complementary with regard to HIREDY and MXSCDX.

- Rules for time: if we only wish to minimize the variable TIME, we can produce similar rules as before by relaxing the assimilation (ASIMDY). In this way, we can accept values for this parameter less than 14.

5 CONCLUSIONS

In this paper a new coding method for evolutionary algorithms in supervised learning is presented. This method converts every attribute domain into a natural number domain. The population will therefore have only natural numbers. The genetic operators (mutation and crossover) are defined in order to work efficiently with this new search space, and they use no convertions from the original attribute domains to the natural number domains, but the evolutionary algorithm works from the begining to the end with natural numbers. As every attribute use one gene in the individual of the population, the size of the search space is decreased, what might allow a faster convergence of the evolutionary algorithm.

The personnel management policies are dominant with respect to the delivery time. When extreme personnel policies (fast or slow) are applied to the organization is difficult to obtain good results for the variables TIME, EFFORT and QUALITY simultaneously. A fast personnel management policies helps to obtain good results for TIME, at the expense of increasing the project EFFORT. With slow policies the opposite occurs. The more significant parameters for finding suitable management policies are in order: assimilation(ASIMDY) and hiring(HIREDY) delay. From the knowledge provided by the rules it is possible to infer that the time delay to transfer people out(TRNSDY) is not significant in most of cases and neither the maximum schedule completion date extension(MXSCDX) for the secenarios analysed in this study.

With respect to the advantages of our evolutionary approach in comparison with C4.5, we would like to note some important issues. Our algorithm searches for rules which cover records with the label identified by the user; C4.5 generates a decision tree in which all the labels appear, and we are definitely not interested in bad projects. From these rules (for bad projects) we can not extract useful knowledge related to management policies. C4.5 always produced more rules covering less records. In many cases, the intervals found by C4.5 for some parameters had a very small range, which is inappropriate if we wish to vary these values in order to achieve a good decision. For example, in the third scenario the three rules produced by C4.5 has very small range for the attribute ASIMDY: [10.22,10.81] for the fist two rules and [9.17,9.27] for the third one. It is obvious that these intervals do not allow the project manager to make a decision.

Acknowledgments

The research was supported by the Spanish Research Agency CICYT under grant TIC1143–C03–02.

References

T. K. Abdel-Hamid. *Software Project Dynamics: an integrated approach.* Prentice-Hall, 1991.

J. S. Aguilar-Ruiz, J.C. Riquelme, and M. Toro. An evolutionary approach to estimating software development projects. *Information and Software Technology*, 14(43):875–882, 2001.

S. Bhattacharyya and G.J. Koehler. An analysis of non–binary genetic algorithms with cardinality 2^v. *Complex Systems*, 8:227–256, 1994.

J. J. Dolado. On the problem of the software cost function. *Information and Software Technology*, 43:61–72, 2001.

U. M. Fayyad and K. B. Irani. Multi-interval discretisation of continuous valued attributes for classification learning. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence.* Morgan Kaufmann, 1993.

G. R. Finnie, G. E. Wittig, and J.-M. Desharnais. A comparison of software effort estimation techniques: using function points with neural networks, case-based reasoning and regression models. *Journal of Systems and Software*, 39(3):281–289, 2000.

R. C. Holte. Very simple classification rules perform well on most commonly used datasets. *Machine learning*, 11:63–91, 1993.

G.J. Koehler, S. Bhattacharyya, and M.D. Vose. General cardinality genetic algorithms. *Evolutionary Computation*, 5(4):439–459, 1998.

T. Mitchell. *Machine Learning.* McGraw Hill, 1997.

J. R. Quinlan. *C4.5: Programs for machine learning.* Morgan Kaufmann, San Mateo, California, 1993.

M. Shepperd and C. Schofield. Estimating software project effort using analogies. *IEEE Transactions on Software Engineering*, 23(12):736–743, 2000.

G. Venturini. Sia: a supervised inductive algorithm with genetic search for learning attributes based concepts. In *Proceedings of European Conference on Machine Learning*, pages 281–296, 1993.

M.D. Vose and A.H. Wright. The simple genetic algorithm and the walsh transform: Part i, theory. *Evolutionary Computation*, 6(3):253–273, 1998.

M.D. Vose and A.H. Wright. The simple genetic algorithm and the walsh transform: Part ii, the inverse. *Evolutionary Computation*, 6(3):275–289, 1998.

F. Walkerden and R. Jeffery. An empirical study of analogy-based software effort estimation. *Empirical Software Engineering*, 42:135–158, 1999.