Available online at www.sciencedirect.com

SCIENCE DIRECT®

European Journal of Operational Research 169 (2006) 477–489

EUROPEAN JOURNAL OF OPERATIONAL RESEARCH

www.elsevier.com/locate/ejor

Solving feature subset selection problem by a Parallel Scatter Search ☆

Félix García López, Miguel García Torres, Belén Melián Batista, José A. Moreno Pérez *, J. Marcos Moreno-Vega

Dpto. de Estadística, Investigación Operativa y Computación, Escuela Técnica Superior de Ingeniería Informática, Universidad de La Laguna, 38271 La Laguna, Spain

Available online 7 October 2004

Abstract

The aim of this paper is to develop a Parallel Scatter Search metaheuristic for solving the Feature Subset Selection Problem in classification. Given a set of instances characterized by several features, the classification problem consists of assigning a class to each instance. Feature Subset Selection Problem selects a relevant subset of features from the initial set in order to classify future instances. We propose two methods for combining solutions in the Scatter Search metaheuristic. These methods provide two sequential algorithms that are compared with a recent Genetic Algorithm and with a parallelization of the Scatter Search. This parallelization is obtained by running simultaneously the two combination methods. Parallel Scatter Search presents better performance than the sequential algorithms.
© 2004 Elsevier B.V. All rights reserved.

Keywords: Scatter Search; Feature subset selection; Parallelization; Metaheuristics

1. Introduction

In a classification problem, the goal is to classify instances that are characterized by a set of features. Then, the class to which each instance belongs is determined. In supervised machine learning, an induction algorithm is typically presented with a set of training instances (examples, cases), where each instance is defined by a vector of features and a class label. The task of the induction algorithm is to induce a classifier that will be used to classify future cases. The classifier is a

☆ This research has been partially supported by the projects TIC2002-04242-C03-01 (70% of which are FEDER founds) and TIC2002-10886E.

* Corresponding author. Tel.: +34 922 318186; fax: +34 922 318170.

E-mail addresses: jamoreno@ull.es (J.A. Moreno Pérez), fcgarcia@ull.es (F. García López), mgarciat@ull.es (M. García Torres), mbmelian@ull.es (B. Melián Batista), jmmoreno@ull.es (J.M. Moreno-Vega).

mapping from the space of feature values to the set of class labels. Since, in practical applications, the set of features can be very large, in order to classify future instances, it is important to select a smaller subset of these features. As pointed in [10], this dimensionality reduction has several advantages: *a reduction in the cost of acquisition of the data*, *improvement of the comprehensibility of the final classification model*, *a faster induction of the final classification model and an improvement in classification accuracy*.

The feature subset selection problem consists of finding a subset of the original set of features, such that an induction algorithm using only these features is able to generate a classifier with the best performance. Selecting the optimal feature subset is an *NP*-hard optimization problem [12]. Therefore exact algorithms should not be used due to the complexity of the problem. For example, determining the optimal binary decision tree is an *NP*-hard problem [9]. Several heuristic algorithms have been proposed for solving the feature subset selection problem. One of the most widely used metaheuristics are the Genetic Algorithms. These algorithms have been proposed and analysed for the feature subset selection problem in [7,17,18,21]. The obtained results show that Genetic Algorithms are appropriate methods for this problem. We propose the use of another evolutive metaheuristic (Scatter Search [13]) to solve this problem and compare one of our proposed Scatter Search procedures with a recent Genetic Algorithms. We have not found any reference about the application of Scatter Search to the feature selection.

Two different approaches for selecting the subset of features can be considered: the wrapper and filter approaches. The filter approach selects the features using a preprocessing step that ignores the induction algorithm. The main disadvantage of this procedure is that it ignores the effect of the subset of features in the induction algorithm. Two filter-based algorithms are RELIEF [11] and FOCUS [3]. The first one assigns a weight to each feature according to its relevance for classifying. To do so it samples several examples randomly and compares the example with the two nearest examples of the same and opposite class. The latter algorithm examines all subsets of features by selecting the minimal subset of features that is sufficient to classify the examples.

In the wrapper approach, the induction algorithm selects the optimal subset of features by itself. Two well known wrapper approaches are forward subset selection (FSS) and backward subset selection (BSS) [5]. FSS starts with an empty subset of features and, at each step, it adds to the subset the feature that most improves the classification. This process is iterated until no improvement is possible. In BSS the initial subset consists of all the available features and, at each step, the worst feature is eliminated from the subset. As in FSS, this process is repeated until no improvement is possible. The Parallel Scatter Search proposed in this paper is based on the wrapper approach.

We consider three paradigms of learning: the *Instance-Based Learning* approach, the *Bayesian Learning procedures*, and the *Decision Tree* methods [14]. The first approach uses the nearest examples to predict the label of the instance, given a set of examples and an instance to be classified. In particular, we use the instance-based algorithm called IB1 [2], that classifies each instance with the label of the nearest example. For the purpose of classifying each instance with the label of the nearest example, IB1 considers all the features, although, in general, only a few of them are highly relevant. The Bayesian Learning algorithms use probability as an approach for classification. The Naive Bayes classifier consists in using Bayes theorem to estimate "a posteriori" probabilities of all possible classifications. For each instance, the classification with the highest "a posteriori" probability is chosen. Decision trees classify instances by testing the instance at each node it reaches. The procedure starts at the root and, at each node, moving down the trees branch according to the result of the test. Leaf nodes give the classification of all instances that reach the leaf. We use the top–down induction decision tree algorithm C4.5 devised by Quinlan [16]. The C4.5 algorithm is an improvement of the classical ID3 (Interactive Dichotomer 3) algorithm for constructing decision trees.

Scatter Search [13] is an evolutionary algorithm in which a set of solutions evolves due to mechanisms of combination between solutions. Unlike other strategies of combination of existing rules like genetic algorithms, the search for a local optimum is a guided task. In order to carry out this strategy, given a population of solutions, a reference set (*RefSet*) is considered. This *RefSet* is generated attempting to intensify and diversify the population of solutions. After combining the solutions in the reference set, a local search procedure is applied to the resulting solution, and the *RefSet* is updated to incorporate both good and disperse solutions. These steps are repeated until a stopping condition is met.

Parallel implementations of metaheuristics appear quite naturally as an effective alternative to speed up the search for approximated solutions of combinatorial optimization problems. We show that they not only allow solving larger problems or finding improved solutions with respect to their sequential counterparts, but also lead to more precise random algorithms. We say that a random algorithm Alg_A is more precise than a random algorithm Alg_B if, after running both algorithms the same number of times, Alg_A reaches objective function values with less standard deviation. In addition, we analyse the effect of running simultaneously several combination strategies by using different processors.

Section 2 describes the feature subset selection problem. The proposed sequential Scatter Search is described in Section 3 and its parallelization in Section 4. Finally, the computational experience and conclusions are shown in Sections 5 and 6, respectively.

2. The feature subset selection problem

Let A be a set of given instances, which are characterized by d features $X = \{X_j : j = 1, \ldots, d\}$. Each feature is either a nominal or a linear attribute. An attribute is linear if the evaluation of the difference between two of its values has sense (being discrete or continuous); otherwise it is nominal. Furthermore, each instance has a label that indicates the class to which it belongs. In order to carry out the task of classifying by means of supervised learning, we consider the subset of instances $T \subset A$ in which labels are known and can be used as training examples, and the subset $V = A \backslash T$ of instances to be classified (validation instances). The labels of V will only be used to measure the performance of the classifier. In the feature subset selection problem, the set of features with the best performance must be obtained. The accuracy percentage is often used to measure the performance of a classifier. Then, the optimization problem associated consists of finding the subset $S \subseteq \{X_j : j = 1, \ldots, d\}$ with higher accuracy percentage. However, this percentage can only be estimated using the validation instances, since V is only a subset of the set of instances to be classified.

The k-fold *cross-validation* method is widely used to estimate the accuracy percentages of a subset of features S on a given set of instances B. The method proceeds in the following way. The set of instances B is randomly divided into k disjoint subsets of equal size $B_1, B_2, \ldots, B_k$ and k trials are carried out. In the trial i, B_i is considered the test set and the training set is the union of the other subsets $T_i = B \backslash B_i$. In each trial, the test instances are classified using the learning algorithm. The estimated accuracy percentage of the classifier is the average of the accuracy percentages over all the trials. The estimated accuracy percentage of a subset of features S on a given set of instances B using *cross-validation* is stated as follows:

$$f_B(S) = 100 \frac{|a \in B : \tilde{c_a} = c_a|}{|B|}, \tag{1}$$

where c_a is the class of each instance a and $\tilde{c_a}$ is the class assigned by the classifier.

In the computational experience we consider, as inductive classifiers, the induction algorithm IB1, the Naive Bayes algorithm, and the C4.5 Decision Tree algorithm provided by the Weka Machine Learning Project [20].

If IB1 is used, for each instance v in the test set, we calculate its nearest example t in the training set and then we consider that both of them belong to the same class and have the same label (i.e., $\tilde{c_v} = c_t$ with c_t the label of t and $\tilde{c_v}$ the label of the class assigned to v). The distance function considered was the heterogeneous Euclidean overlap metric

(HEOM), which can handle both nominal and linear attributes [19]. The overlap metric was applied to nominal attributes, and the normalized Euclidean distance was considered for linear attributes. Let $t = (t_1, t_2, \dots, t_d, c_t)$ be an example with value t_i for the ith feature and label c_t, and let $v = (v_1, v_2, \dots, v_d, c_v)$ be an instance, with similar notation. Let $S \subseteq \{X_j : j = 1, \dots, d\}$ be the feature subset considered. The distance between t and v is defined as

$$\mathrm{dist}_{\mathrm{HEOM}}(t, v) = \sum_{X_j \in S} \mathrm{dist}^2(t_j, v_j)$$

with

$$\mathrm{dist}(t_j, v_j) = \begin{cases} 1 & \text{if } t_j \text{ or } v_j \text{ is unknown}, \\ \mathrm{dist}_s(t_j, v_j) & \text{if } X_j \text{ is nominal}, \\ \mathrm{dist}_r(t_j, v_j) & \text{if } X_j \text{ is linear}, \end{cases}$$

where dist_s is the overlap metric and dist_r is the normalized Euclidean distance. That is

$$\mathrm{dist}_s(t_j, v_j) = \begin{cases} 0 & \text{if } t_j = v_j, \\ 1 & \text{otherwise} \end{cases}$$

and

$$\mathrm{dist}_r(t_j, v_j) = \frac{|t_j - v_j|}{\max_j - \min_j},$$

where $\max_j$ and $\min_j$ are respectively the maximum and minimum values of the feature X_j in the training set. Note that, since the validation set is assumed unknown, the normalized distance can be greater than 1 for instances out of the training set.

The Naive Bayes Classifier is a practical method very appropriated when the attributes that describe the instances are conditionally independent given the classification. Given the attributes $t = (t_1, t_2, \dots, t_d)$ that describe an instance, the most probable class is

$$c_t = \arg\max_{c \in C} P(c | X_1 = t_1, X_2 = t_2, \dots, X_d = t_d).$$

By Bayes theorem

$$c_t = \arg\max_{c \in C} P(X_1 = t_1, \\ X_2 = t_2, \dots, X_d = t_d | c) P(c).$$

Then, assuming the conditional independence, the Naive Bayes classifier is stated as

$$c_t = \arg\max_{c \in C} P(c) \prod_{j=1}^{d} P(X_j = t_j | c).$$

In practical applications the theoretical probabilities are replaced by their estimations. Each probability is estimated by the corresponding frequencies in the training set. One of the two major objections to this method is the case where none of the training instances in a given class have an attribute value. If $P(X_j = t_j | c) = 0$ then every instance with this value cannot be classified in class c. Therefore, modified estimations of these probabilities are used. The other major objection is that the conditional independence assumption is often violated in real applications. However, it works well even in that case because it is only needed that

$$\arg\max_{c \in C} P(X_1 = t_1, \dots, X_d = t_d | c) P(c) \\ = \arg\max_{c \in C} P(c) \prod_{j=1}^{d} P(X_j = t_j | c)$$

and the feature selection procedure helps to choose those attributes that are conditionally independent given the classification.

The C4.5 algorithm is an improvement of the classical ID3 (Interactive Dichotomer 3) method for constructing a decision tree. The improvement includes a method for dealing with numeric attributes, missing values, noisy data, and generating rules for trees. The basic ID3 is a "divide and conquer" method that works as follows. Firstly, it selects an attribute test to place at the root node and make a branch for each possible result of the test. Usually, each test involves only an attribute and one branch is made for each possible value of the attribute. This splits up the training set into subsets and the process is repeated recursively with each branch, using only those instances that actually reach the branch. When all the instances at a node have the same classification, stop developing that part of the tree.

The algorithms determine the test to place at each node. The ID3 uses the information gain criterion to construct the decision tree. The information gain is measured by the purity of the set of instances corresponding to each branch. The purity of a set of instances is measured by the amount of information required to specify the class of one instance that reaches the branch. The use of the gain ratio is one of the improvements that were made to ID3 to obtain the C4.5 algorithm. The gain ratio is a modification of the information gain measure to compensate that it tends to prefer attributes with large number of possible values. The gain ratio takes into account the number and size of the daughter nodes into which a test splits the training set.

For the purpose of guiding the search for the best subset of features (training) and measuring the effectiveness of a particular subset of features after the search algorithm has chosen it as solution of the problem (validation), the function (1) for 2-fold cross-validation is used. To guide the search, $f_T(\cdot)$ is considered and to measure the effectiveness, $f_V(\cdot)$ is used. In validation, we consider 5×2 cross-validation (5×2 cv) [6] that consists of dividing the set V into 2-folds and then conducting two trials. This is done for 5 random arrangements of V. However, in training, we use 1×2 cv, where only one arrangement is done.

3. Application of SS to the feature subset selection problem

The aim of this section is to describe the characteristics of the proposed Scatter Search. Scatter Search (SS) [13] is a population-based metaheuristic that uses a reference set to combine its solutions and construct others. The method generates an initial reference set from a population of solutions. Then, several subsets are selected from this reference set. The solutions of each selected subset are combined to get starting solutions to run an improvement procedure. The result of the improvement can motivate the updating of the reference set and even the updating of the population of solutions. The process is iterated until a stopping condition is met.

The pseudocode of the Sequential Scatter Search is described in Fig. 1. The high level procedures used for developing the Scatter Search are the following.

```
procedure Sequential Scatter Search
begin
    CreatePopulation($Pop$,$PopSize$);
    GenerateReferenceSet($RefSet$, $RefSetSize$);
    repeat
        repeat
            SelectSubset($Subset$, $SubSetSize$);
            CombineSolutions($SubSet$, $CurSol$);
            ImproveSolutions($CurSol$, $ImpSol$);
        until ($StoppingCriterion1$);
        UpdateReferenceSet($RefSet$);
    until ($StoppingCriterion2$);
end.
```

Fig. 1. Sequential Scatter Search metaheuristic pseudocode.

3.1. CreatePopulation

This procedure creates the initial population (Pop), which must be a wide set consisting of disperse and good solutions. Several strategies can be applied to get a population with these properties. The solutions to be included in the population can be created, for instance, by using a random procedure to achieve a certain level of diversity. For the feature subset selection problem, the solution space size depends on the number of features of the problem. Therefore, the size of the initial population is fixed depending on the number of features. We consider $|Pop| = d^2$, where d is the number of features.

In order to build a solution, we use the vector of weights of the features $P(X) = (P(X_1), \ldots, P(X_d))$, given by $P(X_j) = f_T(\{X_j\})$. These weights indicate the quality of the feature for classifying by itself. Let L be the set of features X_j with the highest weights $P(X_j)$. The proposed strategy consists in iteratively selecting at random one of the $|L|$ best possible features (according to P) while its inclusion improves the set. The algorithm is stated in Fig. 2.

3.2. GenerateReferenceSet

A set of good representative solutions of the population is chosen to generate the reference set ($RefSet$). The good solutions are not limited to those with the best objective function values. The considered reference set consists of $RefSetSize1$

```
(1) Set $S = \emptyset$.
(2) Repeat
    (a) Select at random a feature from $L$. Let $X_{j*}$ be the selected feature.
    (b) If $f_T(\{X_{j*}\} \cup S) \geq f_T(S)$ then
            $S \leftarrow S \cup \{X_{j*}\}$ and
            Let $X_j \notin L$ be the feature with the highest $P(X_j)$, then
              $L \leftarrow (L \setminus \{X_{j*}\}) \cup \{X_j\}$.
    until no improvement is reached.
```

Fig. 2. Building strategy.

solutions with the best values of f and *RefSetSize*2 diverse solutions. Then *RefSetSize* = *RefSetSize*1 + *RefSetSize*2. The reference set is generated by selecting first the *RefSetSize*1 best solutions in the population and secondly adding *RefSetSize*2 times the most diverse solution in the population. Let C be the set of features that belong to any solution already in the reference set; i.e.,

$$C = \bigcup_{S \in RefSet} S.$$

The diversity of each solution S is given by the symmetric difference between S and C defined as follows:

$$\mathrm{Div}(S) = \mathrm{Diff}(S, C) = |(S \cup C) \setminus (S \cap C)|.$$

The algorithm proposed to generate the reference set is described in Fig. 3.

3.3. SelectSubset

We consider, as usually in the applications of Scatter Search, all the subsets of two solutions in the current reference set of solutions. The solutions in the subsets are then combined to construct other solutions.

```
(1) Initialize:
    (a) Let $RefSet$ be the empty set.
    (b) Add to $RefSet$ the $RefSetSize1$ best solutions in $Pop$.
    (c) Obtain the initial set of features: $C = \cup_{S \in RefSet} S$.
(2) Repeat:
    (a) For each $S \notin RefSet$, calculate $Div(S, C)$.
    (b) Set $S^* = \arg\max Div(S, C) : S \notin RefSet$.
    (c) $RefSet \leftarrow RefSet \cup S^*$.
    (d) $RefSetSize \leftarrow RefSetSize + 1$.
    (e) Update $C$.
    until $RefSetSize = RefSetSize1 + RefSetSize2$.
```

Fig. 3. Generating the reference set.

3.4. CombineSolutions

The combination procedure tries to combine good characteristics of the selected solutions to get new current solutions. The aim is to get good solutions, which are not similar to those already in the reference set.

We consider two combination methods, which are both greedy strategies. Let S_1 and S_2 be the solutions in the subset. Each combination method generates two new solutions, S'_1 and S'_2. We will refer to the first strategy as greedy combination (GC) and to the second as reduced greedy combination (RGC). They both start by adding to the new solutions S'_1 and S'_2 the features common to S_1 and S_2. Then at each iteration one of the remaining features in S_1 or S_2 is added to S'_1 or S'_2. The reduced version only considers those features that have appeared in good solutions found during the search procedure. The description of the greedy combination (GC) is stated in Fig. 4.

The reduced greedy combination strategy (RGC) differs from the first one in that, instead of considering the whole set of features in $C = (S_1 \cup S_2) \setminus (S_1 \cap S_2)$, it only uses the features with the highest accuracy percentages. The initial set C is reduced by applying the following procedure. Let Q be a weights vector defined in the following way. For each feature $X_j \in C$, $Q(X_j)$ is the average estimated accuracy percentage of all the solutions containing the feature X_j. Then, $Q(X_j)$ is stated as follows:

$$Q(X_j) = \frac{1}{|\{i : X_j \in S_i\}|} \sum_{\{i : X_j \in S_i\}} f(S_i).$$

Let $\overline{Q}$ be the average of the values $Q(X_j)$ such that $X_j \in C$,

```
(1) Initialize new solutions:
      $S_1' \leftarrow S_1 \cap S_2$.
      $S_2' \leftarrow S_1 \cap S_2$.
      Let $C = (S_1 \cup S_2) \setminus (S_1 \cap S_2)$.
(2) Repeat
    (a) For each feature $X_j \in C$ evaluate $f_T(S_1' \cup \{X_j\})$ and $f_T(S_2' \cup \{X_j\})$.
    (b) Let $j_1^*$ and $j_2^*$ be the features such that
            $f_T(S_1' \cup \{X_{j_1^*}\}) = \max_j \{f_T(S_1' \cup \{X_j\})\}$
        and
            $f_T(S_2' \cup \{X_{j_2^*}\}) = \max_j \{f_T(S_2' \cup \{X_j\})\}$
        respectively.
    (c) If $f_T(S_1' \cup \{X_{j_1^*}\}) > f_T(S_1')$ or $f_T(S_2' \cup \{X_{j_2^*}\}) > f_T(S_2')$ then
         (i) If $f_T(S_1' \cup \{X_{j_1^*}\}) > f_T(S_1')$ and $f_T(S_2' \cup \{X_{j_2^*}\}) \le f_T(S_2')$, then
             set $k = 1$.
        (ii) If $f_T(S_1' \cup \{X_{j_1^*}\}) \le f_T(S_1')$ and $f_T(S_2' \cup \{X_{j_2^*}\}) > f_T(S_2')$, then
             set $k = 2$.
       (iii) If $f_T(S_1' \cup \{X_{j_1^*}\}) > f_T(S_1')$ and $f_T(S_2' \cup \{X_{j_2^*}\}) > f_T(S_2')$,
             then set $k = \arg\max_{\{} f_T(S_1' \cup \{X_{j_1^*}\}), f_T(S_2' \cup \{X_{j_2^*}\})\}$. If
             $f_T(S_1' \cup \{X_{j_1^*}\}) = f_T(S_2' \cup \{X_{j_2^*}\})$ then $k$ is the index (1 or
             2) corresponding to the solution with the smaller number of fea-
             tures. If both solutions have the same number of features then
             choose $k$ randomly.
        Add $X_{j_k^*}$ to the solution $S_k'$, set $C = C \setminus X_{j_k^*}$ and go to 2.
    until there is no improvement; i.e., $f_T(S_1' \cup \{X_{j_1^*}\}) \le f_T(S_1')$ and $f_T(S_2' \cup$
    $\{X_{j_2^*}\}) \le f_T(S_2')$
```

Fig. 4. The GC combination.

$$\overline{Q} = \frac{1}{|C|} \sum_{j=0}^{|C|} Q(X_j).$$

The strategy RGC uses the features $X_j \in C$ such that $Q(X_j) \geqslant \overline{Q}$.

3.5. *ImproveSolutions*

The *ImproveSolutions* method is applied every solution, S, generated by the combination method explained above. Let *CA* be the set of features that do not belong to the solution S. Then, the features $X_j \in CA$ are ordered according to their weights $P(X_j)$. The improving method is described in Fig. 5.

The aim of the method is to add to the solution those characteristics that improve it. All the solutions reached by the *ImproveSolutions* method are recorded in a set, *ImpSolSet*, which is then used to update the reference set.

```
(1) Let $X_{(1)}, \ldots, X_{(|CA|)}$ be the features ordered such that
          $P(X_{(j)}) \ge P(X_{(j+1)})$.
(2) $j \leftarrow 0$.
(3) Repeat:
      (a) $j \leftarrow j + 1$.
      (b) If $f_T(S \cup \{X_{(j)}\}) \ge f_T(S)$, then $S \leftarrow S \cup \{X_{(j)}\}$.
    until $(j = |CA|)$
```

Fig. 5. The improving method.

3.6. *UpdateReferenceSet*

Finally, after obtaining all the improved solutions, *RefSet* is updated according to intensity and diversity criteria. First of all, we select the $|RefSet|/2$ best solutions from $RefSet \cup ImpSolSet$. Then, *RefSet* is updated according to the diversity criterium by applying the procedure explained in 3.2.

4. Parallel Scatter Search

Although metaheuristics provide quite effective strategies for finding approximate solutions to combinatorial optimization problems, the computational times associated with the exploration of the solution space may be very large. With the proliferation of parallel computers, parallel implementations of metaheuristics appear quite naturally as an alternative to speedup the search for approximate solutions. Moreover, parallel implementations also allow solving larger problems or finding improved solutions, with respect to their sequential counterparts, due to the partitioning of the search space.

Therefore, parallelism is a possible way not only to reduce the running time of local search algorithms and metaheuristics, but also to improve their effectiveness and precision. The first parallelization of Scatter Search has been proposed by García et al. [8]. The authors considered three parallel strategies in order to reduce the running time of the algorithm and increase the exploration in the solution space. Moreover, some of these strategies improved the quality of the solutions.

In this paper, we consider another straightforward parallel strategy to improve the precision of the Scatter Search metaheuristic without increasing the running time. We can obtain more precise implementations of Scatter Search by using different combination methods and parameter settings at each processor, leading to high quality solutions for different classes of instances of the same problem, without too much effort in parameter tuning and with the same execution time as the sequential algorithm.

Fig. 6 shows the pseudocode of our parallel implementation. Our Parallel Scatter Search (PSS) applies a different combination method (denoted by *CombineSolutions*$_r$) at each processor ($r = 1, \ldots, n_{pr}$). Since we have developed two alternative combination methods, in the computational experience presented in Section 5, $n_{pr} = 2$.

The development of several combination methods for the Scatter Search metaheuristic has been utilized in previous works. For example, Campos et al. [4] designed different combination methods for a sequential implementation of Scatter Search for the linear ordering problem. They also assessed the relative contribution of each method to the quality of the final solution. Based on the results obtained, they used the combination method that presented a better performance. However, in this paper we run two combination methods simultaneously by using two processors. Scatter search requires high computational times, complicating the sequential execution of several consecutive combination methods. The goal of the proposed parallelization is to achieve an improvement of the quality of the solution, using the same computational time used by the sequential algorithm.

```
procedure Parallel Scatter Search
begin
    CreatePopulation($Pop$, $PopSize$);
    GenerateReferenceSet($RefSet$, $RefSetSize$);
    repeat
        repeat
            SelectSubSet($SubSet$, $SubSetSize$);
            for each processor $r = 1, \ldots, n_{pr}$ do in parallel
            begin
                $CombineSolutions_r$($SubSet$, $CurSol_r$);
                ImproveSolutions($CurSol_r$, $ImpSol_r$);
            end;
        until ($StoppingCriterion1$);
        UpdateReferenceSet($RefSet$);
    until ($StoppingCriterion2$);
end.
```

Fig. 6. Parallel Scatter Search pseudocode.

5. Computational results

The objective of the computational experiments is to show the performance of the Scatter Search in searching for a reduced set of features with high accuracy. Firstly, we compared a Scatter Search with a Genetic Algorithm using three standard classifiers (IB1, Naive Bayes and C4.5). The data showed a superiority of the Scatter Search over the Genetic Algorithm. Moreover, the computational experience carried out corroborates that the comparisons of the Scatter Search metaheuristic and the Genetic Algorithm is similar when using any of these classifiers. This means that any of the three considered classifiers (IB1, Naive Bayes and C4.5) can be used to analyse the efficiency of these metaheuristics. Then, we designed an experiment to find the suitable values for the parameters of our Scatter Search implementation. Finally, we compared both Sequential Scatter Search algorithms with the parallel procedure using the parameters obtained.

The datasets considered in our computational experiments were obtained from the UCI repository [15], from which full documentation about all datasets can be obtained. We chose them taking into account their size and use in machine learning research. The selected datasets have more than 300 instances because small datasets can motivate overfitting. An induction algorithm overfits the dataset if it models the training examples too well and its predictions are poor. Table 1 summarises the characteristics of the chosen datasets. The first two columns correspond to the name of the datasets as it appears in the UCI repository and the identifier (*Id*) used in forthcoming tables. The intermediate three columns show the total number of features, the number of nominal features and the number of (numerical) linear features. Finally, the last two columns summarise the number of instances and classes in the dataset.

We use the 5×2 cross-validation method to measure the accuracy percentage of the resulting subset of features selected by the algorithms. However, in order to increase the efficiency of the search, during the learning process only 1×2 cross-validation was considered. To perform the comparisons between classifiers we used the *F*-test (see [1]) at statistical significance level of 0.95. The *F*-test is a better test than the classical two-tailed *t*-test for comparing classifiers.

In order to analyse the influence of the classifiers in the performance of the metaheuristic algorithms, we solved several problems with both algorithms using the IB1, Naive Bayes and C4.5 classifiers. Table 2 shows the results obtained with Sequential Scatter Search Greedy Combination (SSS-GC) and Genetic Algorithm. The data shown in the table correspond to values of the parameters $|L| = d/2$ and $|RefSet| = 5$ in the SSS-GC. The Genetic Algorithm applied is that provided by Weka [20] with rank-based selection strategy and the parameter setting proposed by Yang and Honavar [21]. Namely, the values of the parameters are population size 50, number of generations 20, probability of cross-over 0.6, probability of mutation 0.001 and probability of selection of the highest ranked individual 0.6. For each method, we show average accuracy percentages, average number of features and standard deviations over 10 runs for each dataset. We also report the average results over all the datasets considered.

The results shown in this table are similar for both metaheuristics. In both cases, the classifier that selects the smallest number of features is C4.5 and IB1 provides the smallest reduction in the set of features. The classifier that shows the best percentages is Naive Bayes, but the differences are not significant (at 95% level). These are known

Table 1
Summary of general characteristics of datasets

DataBase	Id	Features			Instances	Classes
		All	Nom	Lin		
Heart (Cleveland)	*HC*	13	7	6	303	2
Soybean Large	*SbL*	35	29	6	307	19
Vowel	*Vw*	10	0	10	528	11
Credit Screening	*Cx*	15	9	6	690	2
PimaIndianDiabetes	*Pm*	8	0	8	768	2
Anneal	*An*	38	29	9	798	5
Thyroid (Allbp)	*TAb*	28	22	6	2800	2
Thyroid (Sick-Euthyroid)	*TSE*	25	18	7	3163	2
BreastCancer	*BC*	9	0	9	699	2
Ionosphere	*Io*	34	0	34	351	2
HorseColic	*HoC*	21	14	7	368	2
WisconsinBreastCancer	*WBC*	30	0	30	569	2

Table 2
Accuracy and number of features for each classifier with the Genetic Algorithm (GA) and the Scatter Search (SS)

	IB1		Naive–Bayes		C4.5	
	Accuracy	Features	Accuracy	Features	Accuracy	Features
Scatter Search (SSS-GC)						
Cx	80.67 ± 2.10	6.60 ± 1.43	85.30 ± 1.41	5.10 ± 2.28	83.94 ± 2.17	4.7 ± 1.89
Hoc	81.63 ± 3.26	7.20 ± 2.25	82.83 ± 3.06	4.60 ± 0.70	83.75 ± 3.12	4.00 ± 1.49
Io	88.88 ± 3.30	6.50 ± 2.92	90.36 ± 3.16	8.30 ± 2.58	87.07 ± 3.81	5.00 ± 2.62
An	95.72 ± 1.71	11.40 ± 1.78	92.36 ± 2.43	9.40 ± 2.27	91.45 ± 2.19	8.40 ± 1.35
	86.72	7.92	87.71	6.85	86.54	5.25
Genetic Algorithm (GA)						
Cx	79.94 ± 2.87	7.30 ± 1.83	85.04 ± 1.05	6.50 ± 1.65	83.51 ± 1.26	5.60 ± 1.17
Hoc	80.49 ± 2.69	11.30 ± 2.83	82.01 ± 2.48	7.20 ± 2.25	83.86 ± 2.33	5.40 ± 1.71
Io	86.26 ± 2.94	9.60 ± 3.89	90.77 ± 1.59	12.70 ± 2.63	88.43 ± 2.19	10.80 ± 2.39
An	95.75 ± 1.54	18.90 ± 2.85	94.21 ± 1.22	20.70 ± 2.00	92.43 ± 2.03	15.90 ± 3.25
	85.61	11.77	88.00	11.77	87.08	9.42

properties of the classifier and do not depend on the subset of selected features. Therefore, the conclusions would be similar using any classifier. We use only the IB1 classifier to carry out the remaining experiments.

Note that the results in Table 2 corroborate that Sequential Scatter Search Greedy Combination (SSS-GC) has a better performance than Genetic Algorithm, getting always a significantly smaller number of features for each classifier (at 95% level). The same results were achieved when using Sequential Scatter Search Reduced Greedy Combination (SSS-RGC).

Table 3 summarises the results obtained by modifying the values of some of the key parameters of the Sequential Scatter Search procedures. The considered parameters are the size of the reference set, $|RefSet|$, and the size of the list, $|L|$, used in the constructive phase. We fixed $|L| = 3$, $d/2$ and $|RefSet| = 5$, 10, if $|L| = 3$, and $|RefSet| = 5$, 10, d^2, if $|L| = d/2$. The application of the F-test over the results in Table 3 corroborate that there is no significative difference (at 95% level) between the percentages for the different parameter combinations. However, the number of features selected is significantly smaller (at 95% level) for $|RefSet| = d^2$. Therefore, in the rest of the computational experiments, we only consider $|L| = d/2$ and $|RefSet| = d^2$.

Table 4 shows a comparison between the accuracy of the Sequential Scatter Searches (SSS-GC and SSS-RGC) and the Parallel Scatter Search (PSS) using the IB1 classifier. For each method, we provide average accuracy percentages and standard deviations over 10 runs for each dataset. The first average and standard deviation for each dataset show the results obtained using all the features. We also report, at the bottom, average results over all the datasets considered. From the results obtained, the following observations can be made. First of all, IB1 with all the features provides the best accuracy percentages, but the difference is not significant (at 95% level) in most datasets. Secondly, PSS has higher precision than both SSS-GC and SSS-RGC.

Finally, we analyse the number of features selected by the algorithms. Table 5 shows the total number of features of each dataset and the average number of features selected with each algorithm and their standard deviations. At the bottom we give the average of these numbers over all the datasets considered and the reduction percentages. Both Sequential Scatter Search procedures have a similar behavior. However, PSS uses a smaller number of features and its standard deviation is the lowest of all the considered algorithms. Moreover, the parallel SS reduces significantly (at 95% level) the set of features selected by each sequential SS to classify for some datasets. For example, for TSE dataset, the number of features is reduced from 5.10 to 1.90 in average. Considering the number of features of the best solution obtained by

Table 3
Accuracy and number of features with the several values of the parameters for both combinations

$\|RefSet\|$	$\|L\| = 3$		$\|L\| = d/2$		
	5	10	5	10	d^2
Sequential Scatter Search with Greedy Combination					
	Accuracy				
Cx	78.64 ± 2.46	79.82 ± 3.33	80.67 ± 2.10	79.71 ± 2.82	83.28 ± 3.12
Hoc	81.03 ± 3.66	81.85 ± 3.24	81.63 ± 3.26	80.50 ± 4.04	76.69 ± 3.49
Io	89.17 ± 2.60	87.18 ± 2.12	88.88 ± 3.30	87.86 ± 1.86	87.75 ± 1.37
An	94.41 ± 4.88	94.68 ± 4.75	95.72 ± 1.71	95.63 ± 1.91	94.14 ± 3.16
	85.80	85.88	86.82	85.91	85.46
	Number of features				
Cx	6.20 ± 0.79	6.60 ± 1.07	6.60 ± 1.43	6.50 ± 1.65	3.40 ± 1.43
Hoc	7.30 ± 1.49	8.80 ± 2.20	7.20 ± 2.25	8.30 ± 2.06	7.40 ± 2.41
Io	6.90 ± 2.28	7.00 ± 2.36	6.50 ± 2.92	7.10 ± 1.80	6.10 ± 1.37
An	9.90 ± 2.47	10.50 ± 2.17	11.40 ± 1.78	12.70 ± 2.21	8.90 ± 2.89
	7.57	8.22	7.92	8.65	6.45
Sequential Scatter Search with Reduced Greedy Combination					
	Accuracy				
Cx	80.46 ± 2.83	79.51 ± 3.30	80.35 ± 2.23	79.68 ± 3.44	83.91 ± 3.27
Hoc	80.71 ± 3.50	80.71 ± 3.53	78.59 ± 6.16	79.51 ± 3.94	77.94 ± 2.96
Io	87.98 ± 2.78	88.03 ± 2.47	87.23 ± 2.12	87.24 ± 2.37	87.12 ± 1.24
An	94.25 ± 4.84	95.14 ± 3.05	88.78 ± 16.63	95.86 ± 2.37	92.98 ± 2.68
	85.85	85.84	83.73	85.57	85.48
	Number of features				
Cx	6.80 ± 1.32	7.10 ± 1.29	6.60 ± 3.12	6.80 ± 1.40	4.50 ± 2.27
Hoc	8.30 ± 1.34	8.40 ± 1.71	7.30 ± 1.89	7.90 ± 1.73	6.30 ± 2.16
Io	6.00 ± 1.83	6.90 ± 2.47	5.40 ± 2.17	7.20 ± 1.14	5.70 ± 1.06
An	9.90 ± 2.51	10.80 ± 2.39	10.20 ± 3.16	11.60 ± 1.40	8.20 ± 2.66
	7.75	8.30	7.37	8.37	6.17

Table 4
Accuracy percentage and standard deviation in validation

Id	All	SSS-GC	SSS-RGC	PSS
HC	75.98 ± 3.10	74.99 ± 5.31	74.99 ± 5.68	74.91 ± 2.85
SbL	85.02 ± 4.18	82.41 ± 3.49	83.65 ± 3.65	80.53 ± 1.92
Vw	95.29 ± 1.66	93.58 ± 1.39	93.58 ± 1.39	93.64 ± 1.34
Cx	81.54 ± 2.31	83.28 ± 3.12	83.91 ± 3.27	83.39 ± 2.74
Pm	69.71 ± 2.68	67.92 ± 2.35	67.66 ± 2.42	68.10 ± 2.43
An	93.59 ± 1.09	94.14 ± 3.16	92.98 ± 2.68	91.49 ± 2.19
TAb	95.86 ± 0.24	95.53 ± 0.33	95.44 ± 0.44	95.44 ± 0.43
TSE	92.67 ± 0.68	95.09 ± 2.76	95.12 ± 2.78	93.58 ± 2.20
BC	95.48 ± 0.70	95.22 ± 1.07	94.88 ± 1.45	95.11 ± 0.90
Io	85.75 ± 1.30	87.75 ± 1.37	87.12 ± 1.24	87.35 ± 1.56
HoC	75.60 ± 1.99	76.69 ± 3.49	77.94 ± 2.96	76.96 ± 3.79
WBC	95.61 ± 0.82	94.66 ± 1.51	93.57 ± 2.23	93.67 ± 2.36
Average	86.84 ± 1.73	86.77 ± 2.45	86.74 ± 2.52	86.18 ± 2.06

Table 5
Number of features selected for each algorithm

Id	All	SSS-GC	SSS-RGC	PSS
HC	13	6.30 ± 1.64	6.20 ± 2.10	5.56 ± 1.60
SbL	35	15.0 ± 2.71	16.5 ± 2.22	12.80 ± 1.81
Vw	10	7.70 ± 0.68	7.70 ± 0.68	8.00 ± 0.94
Cx	15	3.40 ± 1.43	4.50 ± 2.27	2.80 ± 2.62
Pm	8	4.10 ± 0.99	4.00 ± 0.94	4.20 ± 1.14
An	38	8.90 ± 2.89	8.20 ± 2.66	6.30 ± 2.06
TAb	29	2.80 ± 1.48	2.70 ± 1.83	2.00 ± 1.05
TSE	25	5.10 ± 2.47	5.10 ± 2.42	1.90 ± 1.20
BC	9	5.20 ± 1.62	4.78 ± 1.48	5.40 ± 1.71
Io	34	6.10 ± 1.37	5.70 ± 1.06	3.90 ± 0.88
HoC	21	7.40 ± 2.41	6.30 ± 2.16	4.50 ± 1.51
WBC	30	6.80 ± 2.53	5.50 ± 1.43	6.00 ± 2.63
Average	22.25	6.57 ± 1.85	6.43 ± 1.77	5.28 ± 1.60
Reduction	100.00%	70.47%	71.10%	76.27%

each algorithm, we conclude that PSS is the algorithm that performs better.

6. Conclusions

In this paper, we propose a Scatter Search metaheuristic for solving the Feature Subset Selection Problem. We develop two combination methods: the Greedy Combination and the Reduced Greedy Combination. For the purpose of using both combination methods simultaneously and increasing the exploration of the solutions space, a parallelization of the Scatter Search is developed. The parallelization consists of running each combination method at a different processor.

The obtained computational results corroborate the effectiveness of our parallelization. The parallel SS achieves values of accuracy percentage similar to both sequential SS algorithms, but uses a smaller subset of features. Moreover, the parallel algorithm is more precise than sequential algorithms.

References

[1] E. Alpaydin, Combined 5×2 cv F test for comparing supervised classification learning algorithms, Neural Computation 11 (1999) 1885–1892.

[2] D.W. Aha, D.K. Amd, M.K. Albert, Instanced-based learning algorithms, Machine Learning 6 (1991) 37–66.

[3] J.R. Anderson, M. Matessa, Explorations of an incremental, Bayesian algorithm for categorization, Machine Learning 9 (1992) 275–308.

[4] V. Campos, F. Glover, M. Laguna, R. Martí, An experimental evaluation of a scatter search for the linear ordering problem, Journal of Global Optimization 21 (2001) 397–414.

[5] P. Devijver, J. Kittler, Pattern Recognition: A Statistical Approach, Prentice Hall, 1982.

[6] T.G. Dietterich, Approximate statistical test for comparing supervised classification learning algorithms, Neural Computation 10 (7) (1998) 1895–1923.

[7] F. Ferri, V. Kadirkamanathan, J. Kittler, Feature subset search using genetic algorithm, in: IEE/IEEE Workshop on Natural Algorithms in Signal Processing, IEE Press, 1993, p. Essex.

[8] F. García-López, B. Melián-Batista, J. Moreno-Pérez, J.M. Moreno-Vega, Parallelization of the scatter search for the p-median problem, Parallel Computing 29 (2003) 575–589.

[9] L. Hyafil, R.L. Rivest, Constructing optimal binary decision trees is *np*-complete, Information Processing Letters 5 (1) (1976) 15–17.

[10] I. Inza, P. Larrañaga, R. Etxeberria, B. Sierra, Feature subset selection by Bayesian networks based optimization, Artificial Intelligence 123 (2000) 157–184.

[11] K. Kira, L. Rendell, The feature selection problem: Traditional methods and a new algorithm, in: In 10th National Conference on Artificial Intelligence (AAAI-92), MIT, 1992, pp. 129–134.

[12] R. Kohavi, G.H. John, Wrappers for feature subset selection, Artificial Intelligence 97 (1–2) (1997) 273–324.

[13] M. Laguna, R. Martí, Scatter Search: Methodology and Implementations in C, Kluwer Academic Press, 2003.

[14] T. Mitchell, Machine Learning, Series in Computer Science, McGraw-Hill, 1997.

[15] P.M. Murphy, D.W. Aha, Uci repository of machine learning. URL http://www.ics.uci.edu/mlearn/MLRepository.html.

[16] J. Quinlan, C4.5: Programs for Machine Learning, Morgan Kaufmann, 1993.

[17] W. Siedlicki, J. Sklansky, A note on genetic algorithm for large-scale feature selection, Pattern Recognition Letters 10 (1989) 335–347.

[18] H. Vafaie, K.D. Jong, Robust feature selection algorithmsProceedings of the Fifth IEEE International Conference on Tools for Artificial Intelligence, IEE Press, 1993, pp. 356–363.

[19] D.R. Wilson, T.R. Matinez, Improved heterogeneous distance functions, Journal of Artificial Intelligence Research 6 (1997) 1–34.

[20] I. Witten, E. Frank, Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations, Morgan Kaufmann, 2000.

[21] J. Yang, V. Honavar, Feature Subset Selection using a Genetic Algorithm. Genetic Programming 1997: Proceeding of the Second Annual Conference, Morgan Kaufmann, 1997.