Mach Learn (2010) 78: 381–420
DOI 10.1007/s10994-009-5161-3

A cooperative coevolutionary algorithm for instance selection for instance-based learning

Nicolás García-Pedrajas · Juan Antonio Romero del Castillo · Domingo Ortiz-Boyer

Received: 30 October 2006 / Revised: 10 August 2009 / Accepted: 24 October 2009 /
Published online: 8 December 2009
© The Author(s) 2009

Abstract This paper presents a cooperative evolutionary approach for the problem of instance selection for instance based learning. The model presented takes advantage of one of the recent paradigms in the field of evolutionary computation: cooperative coevolution. This paradigm is based on a similar approach to the philosophy of *divide and conquer*. In our method, the training set is divided into several subsets that are searched independently. A population of global solutions relates the search in different subsets and keeps track of the best combinations obtained. The proposed model has the advantage over standard methods in that it does not rely on any specific distance metric or classifier algorithm. Additionally, the fitness function of the individuals considers both storage requirements and classification accuracy, and the user can balance both objectives depending on his/her specific needs, assigning different weights to each one of these two terms. The method also shows good scalability when applied to large datasets.

The proposed model is favorably compared with some of the most successful standard algorithms, IB3, ICF and DROP3, with a genetic algorithm using CHC method, and with four recent methods of instance selection, MSS, entropy-based instance selection, IMOEA and LVQPRU. The comparison shows a clear advantage of the proposed algorithm in terms of storage requirements, and is, at least, as good as any of the other methods in terms of testing error. A large set of 50 problems from the UCI Machine Learning Repository is used for the comparison. Additionally, a study of the effect of instance label noise is carried out, showing the robustness of the proposed algorithm.

The major contribution of our work is showing that cooperative coevolution can be used to tackle large problems taking advantage of its inherently modular nature. We show that a

Editor: Risto Miikkulainen.

N. García-Pedrajas (✉) · J.A. Romero del Castillo · D. Ortiz-Boyer
Department of Computing and Numerical Analysis, University of Córdoba, Campus Universitario de Rabanales, 14071 Córdoba, Spain
e-mail: npedrajas@uco.es

J.A. Romero del Castillo
e-mail: aromero@uco.es

D. Ortiz-Boyer
e-mail: dortiz@uco.es

combination of cooperative coevolution together with the principle of divide-and-conquer can be very effective both in terms of improving performance and in reducing computational cost.

Keywords Instance selection · Evolutionary algorithms

1 Introduction

The overwhelming amount of data that is available nowadays in any field of research poses new problems for data mining and knowledge discovery methods. This huge amount of data makes most of the existing algorithms inapplicable to many real-world problems. Two approaches have been used to face this problem: scaling up data mining algorithms (Provost and Kolluri 1999) and data reduction. Nevertheless, scaling up a certain algorithm is not always feasible. Data reduction consists of removing from the data missing, redundant, information-poor data and/or erroneous data to get a tractable problem size. Data reduction techniques use different approaches: feature selection (Liu and Motoda 1998), feature-value discretization (Hussain et al. 1999), and instance selection (Blum and Langley 1997). This paper deals with instance selection for instance based learning.

Instance selection (Liu and Motoda 2002) consists of choosing a subset of the total available data to achieve the original purpose of the data mining application as if the whole data is used. Different variants of instance selection exist. Many of the approaches are based on some form of sampling (Cochran 1977; Kivinen and Mannila 1994). However, there are other methods that are based on different principles.

Our aim is focused on instance selection for instance-based learning. We can distinguish two main models (Cano et al. 2003): instance selection as a method for prototype selection for algorithms based on prototypes (such as k-Nearest Neighbors) and instance selection for obtaining the training set for a learning algorithm that uses this training set (such as classification trees or neural networks). Although the proposed model is used for the former approach, it can be used for the latter without any significant modification.

The problem of instance selection for instance based learning can be defined as Brighton and Mellish (2002) "the isolation of the smallest set of instances that enable us to predict the class of a query instance with the same (or higher) accuracy than the original set".

It has been shown that different groups of learning algorithms need different instance selectors in order to suit their learning/search bias (Brodley 1995). This may render many instance selection algorithms useless, if their philosophy of design is not suitable for the problem at hand. Our algorithm does not assume any structure of the data or any behavior of the classifier, adapting the instance selection to the performance of the classifier.

Evolutionary Computation (EC) (Holland 1975; Goldberg 1989; Michalewicz 1994) is a set of global optimization techniques that have been widely used in the last few years for almost every problem within the field of Artificial Intelligence. In evolutionary computation a population (set) of individuals (solutions to the problem faced) are codified following a code similar to the genetic code of plants and animals. This population of solutions is evolved (modified) over a certain number of generations (iterations) until the defined stop criterion is fulfilled. Each individual is assigned a real value that measures its ability to solve the problem, which is called its *fitness*.

In each iteration new solutions are obtained combining two or more individuals (crossover operator) or randomly modifying one individual (mutation operator). After applying these two operators a subset of individuals is selected to survive to the next generation,

either by sampling the current individuals with a probability proportional to their fitness, or by selecting the best ones (elitism). The repeated processes of crossover, mutation and selection are able to obtain increasingly better solutions for many problems of Artificial Intelligence.

Brighton and Mellish (2002) argued that the structure of the classes formed by the instances can be very different, thus, an instance selection algorithm can have a good performance in one problem and be very inefficient in another. They state that the instance selection algorithm must gain some insight into the structure of the classes to perform an efficient instance selection. However, this insight is usually not available or very difficult to acquire, especially in real-world problems with many variables and complex boundaries between the classes. In such a situation, an approach based on EC may be of help. The approaches based on EC do not assume any special form of the space, the classes or the boundaries between the classes, they are only guided by the ability of each solution to solve the task. In this way, the algorithm learns the relevant instances from the data without imposing any constraint in the form of classes or boundaries between them.

Within the field of EC there is a paradigm that focuses on approaching complex tasks by dividing them into simpler subproblems: Cooperative Coevolution (CC). The term was introduced by Potter and De Jong (1994). The basic idea underlying CC is the evolution of partial solutions to complex problems that can cooperate to make up a global solution (Potter and De Jong 1994, 2000; Moriarty and Miikkulainen 1996). In CC each individual is not the solution to a problem, but just a partial solution. Different individuals must be combined to obtain a solution. In this way, modularity is an intrinsic aspect of the methodology. The proposed method for instance selection for instance based learning is based on cooperative coevolution. In this way, our model divides the problem of instance selection into different subproblems, easier to be efficiently solved.

In a first approach our method shares some of the ideas underlying *stratified random sampling* (Liu and Motoda 2002). In stratified random sampling a set of n instances is divided into k non-overlapping subsets of sizes $n_1, n_2, \ldots, n_k$, where $\sum_i n_i = n$. Each subset is called a *stratum*. Then a random sample is extracted from each stratum. In our approach the set of instances is stratified and each stratum is assigned to evolve following a genetic algorithm. In this way, the initial populations are constructed by stratified random sampling. The evolution of the individuals in each population optimizes the classification and storage requirements within the stratum. The main contribution of our method is the use of cooperative coevolution to promote collaboration among the strata. In this way, another population is created that keeps track of the best combination of individuals so far and enforces cooperation among the individuals that evolve using instances of each stratum. The model allows interaction among the simpler searches that are carried out in each stratum, instead of performing a large search in the whole set.

One of the ways of solving complex problems is decomposing them into several simpler tasks, that can be dealt with separately. In instance selection this is problematic. Although we can divide the training set into several subsets, it is difficult to achieve a real problem decomposition, as in order to obtain the nearest neighbor of a given instance, the whole training set must be considered. We can use a stratified approach (Cano et al. 2003) dividing the dataset into several disjoint subsets, and then apply a instance selection algorithm to each subset. However, in such a method, the solution obtained by each algorithm is contaminated by the partial view of the dataset it has. Our model is a way of achieving at least a partial problem decomposition. Several subpopulations evolve taking only into account the instances of a certain subset. The use of another population that combines the results of each subpopulation is able to give the model the necessary global view to accomplish its task.

As in any other field of application, a new algorithm is aimed at improving the results of the existing methods. Thus, our first objective is to obtain better results than state-of-the-art instance selection methods. However, within the field of instance selection, scalability is an important issue. Thus, the proposed model is also intended to be more scalable than other proposals based on genetic algorithms. Our way of approaching scalability is by means of the cooperative coevolution paradigm, as we have stated. So, one of the primary concerns of this work is to study the viability of cooperative coevolution in the field of instance selection. As this paradigm has proven useful for cases where the decomposition of a problem in simpler tasks is interesting, it can be useful in instance selection. This paradigm presents the advantage of offering the possibility of a distributed implementation as a way to achieve scalability for large problems. Due to the computational cost of evolutionary algorithms, it is difficult to apply evolutionary based instance selection algorithms to large problems. We think that the most straightforward way of achieving scalability is through the design of algorithms that can be distributed. In this way, as our basic algorithm can be parallelized but not efficiently distributed, we present a second version that can be both parallelized and distributed. This second version is shown to have similar performance to the original algorithm.

In fact, we are trying to achieve problem decomposition using cooperative coevolution. This problem decomposition is partial, because the problem itself is global, the usefulness of an instance cannot be fully assured without taking into account the whole training set. Here is where cooperative coevolution becomes relevant. The evolution of subcomponents allows for partial problem decomposition, and the possibility of facing simpler problems. The evolution of combinations of subcomponents allows the global view that is needed for an effective instance selection method.

Finally, to avoid confusion, we must remark that the term coevolution alone is usually used for both competitive and cooperative coevolution. However, the two paradigms are quite different. Our model is based on cooperative coevolution, that is, the use of cooperative subcomponents that must work together to solve a complex task. On the other hand, competitive coevolution works with individuals that evolve together and compete for the resources. These individuals usually represent views of the problem and the solution. They can take the form of host/parasites (Hillis 1991), predators/preys (Rosin and Belew 1997), antibodies/antigens (García-Pedrajas and Fyfe 2007), or some more recent methods (Bongard and Lipson 2005a, 2005b). It is also commonly used in the evolution of game players (Moriarty and Miikkulainen 1995; Chellapilla and Fogel 1999). This work is only concerned with cooperative coevolution.

This paper is organized as follows: Section 2 reviews some related work; Section 3 presents the proposed model for instance selection based on cooperative coevolution; Section 4 states the experimental setup; Section 5 shows the results of the experiments; Section 6 shows an additional comparison with some recent algorithms; Section 7 studies the behavior of our method in large datasets; Section 8 discusses the results of our experiments; Section 9 summarizes future research lines; and finally Sect. 10 states the main conclusions of our work.

2 Related work

Instance selection methods have been being developed for many years (Cochran 1977; Kivinen and Mannila 1994). Revisions of most current standard methods can be found in Liu and Motoda (2002) and Brighton and Mellish (2002). These methods are usually designed for nearest neighbors classifiers, taking into account the behavior of these classifiers.

As an alternative to these standard methods, genetic algorithms have been applied for instance selection, considering this task to be a search problem. The application is easy and straightforward. Each individual is a binary vector that codes a certain sample of the training set. The evaluation is usually made considering both data reduction and classification accuracy. Examples of applications of genetic algorithms to instance selection can be found in Kuncheva (1995), Ishibuchi and Nakashima (2000) and Reeves and Bush (2001). We can also mention a first attempt of using competitive coevolution by Hillis (1991), who used a parasite model to reduce the number of instances needed to test a sorting network. That method was able to discard easily solved problems allowing a faster evaluation of the individuals in a genetic algorithm.

One of the most interesting advantages of the application of evolutionary computation to instance selection is that evolutionary approaches do not depend on specific classifiers, and can be used with any instance based classifier. This is in contrast with most standard instance selection algorithms that are specifically designed for k-NN classifiers. For instance, Reeves and Bush (2001) used a genetic algorithm to select instances for RBF neural networks.

Cano et al. (2003) performed a comprehensive comparison of the performance of different evolutionary algorithms for instance selection. They compared a generational genetic algorithm (Goldberg 1989), a steady-state genetic algorithm (Whitley 1989), a CHC genetic algorithm (Eshelman 1990), and a population based incremental learning algorithm (Baluja 1994). They found that evolutionary based methods were able to outperform classical algorithms in both classification accuracy and data reduction. Among the evolutionary algorithms, CHC was able to achieve the best overall performance.

The major problem addressed when applying genetic algorithms to instance selection is the scaling of the algorithm. As the number of instances grows, the time needed for the genetic algorithm to reach a good solution increases exponentially, making it totally useless for large problems. Recently, Cano et al. (2005, 2007) have proposed a stratified approach to alleviate this difficulty.

To the best of our knowledge there is no previous application of cooperative coevolution to instance selection for instance based learning.

3 Instance selection by cooperative coevolution

In cooperative coevolution a number of species are evolved together. Cooperation among individuals is encouraged by rewarding the individuals for their joint effort to solve a target problem. The work in this paradigm has shown that cooperative coevolutionary models present many interesting features, such as specialization through genetic isolation, testing and efficiency (Potter and De Jong 2000). Cooperative coevolution approaches the design of modular systems in a natural way, as the modularity is part of the model. Other models need some *a priori* knowledge to decompose the problem *by hand*. In many cases, either this knowledge is not available or it is not clear how to decompose the problem. The cooperative coevolutionary model offers a very natural way for modeling the evolution of cooperative parts.

Our model is inspired on a stratified approach. It is obvious that if we could perform independent searches within each stratum, the task would be simplified. However, this is not possible as the nearest neighbors of each instance can be in any stratum. Considering only the instances in the stratum will yield to erroneous evaluation of the neighbors. So, we develop a model that combines a search in independent strata together with a method of combining those independent searches in such a way that the results are useful and efficient.

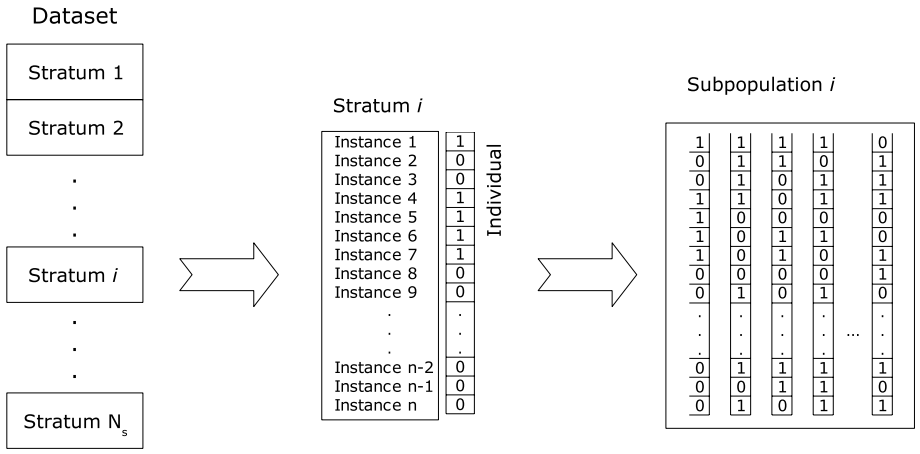


Fig. 1 Distribution of the dataset among the different subpopulations of instances and representation of an individual

Our cooperative model, called COOPERATIVE COEVOLUTIONARY INSTANCE SELECTION (CCIS) algorithm, is based on two separate populations that evolve cooperatively.[1] These two populations are:

– *Population of selectors.* This population is made up of s independent subpopulations. The whole training set is divided into s approximately equal parts and each part is assigned to a subpopulation. Each individual of a subpopulation is a subset of instances for the corresponding subset of training instances (see Fig. 1). Every subpopulation is evolved using a standard genetic algorithm. The individuals of these subpopulations will be called throughout the paper *selectors*. We will use the term *population of selectors* when referring to all the subpopulations of selectors as a whole.

  The subpopulations in the population of selectors are also evolved separately, without exchanging genetic material. Thus, we can consider that our proposal evolves $s + 1$ separate populations. To avoid confusion, as the two populations have different individuals, we will consider in the following that we evolve two separate populations, one of them made up of s independent subpopulations of selectors.

– *Population of combinations of instances sets.* Each member of the population of combinations is the combination of an individual from every subpopulation.

  The population of combinations keeps track of the best combinations of selectors for different subsets of instances, selecting the combinations that are promising for the final global selector selection of the whole dataset. The individuals of this population will be called throughout the paper *combinations*.

The individuals of the populations of selectors are subject to two operations: crossover and mutation. The crossover operator is the *Half Uniform Crossover* (HUX) (Eshelman 1990). This operator generates two offspring from two parents. Each offspring inherits the matching bits of the two parents, and half of the non-matching bits from each parent alternately.

[1] A model sharing some of these basic ideas has already been successfully applied to the evolution of modular neural networks (García-Pedrajas et al. 2002) and ensembles of neural networks (García-Pedrajas et al. 2005).

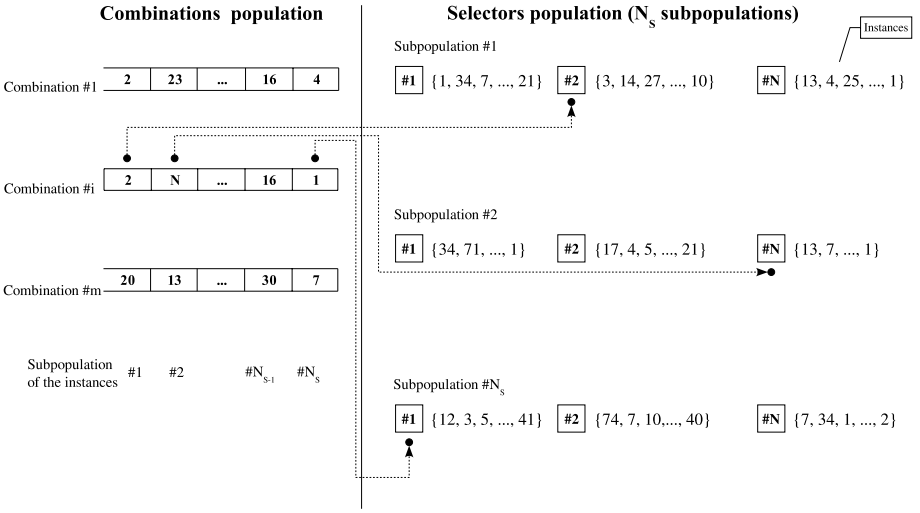


Fig. 2 Populations of selectors and combinations. Each element of the population of combinations is a reference to an individual of the corresponding subpopulation of selectors

Mutation operator takes two forms: random mutation and local search mutation. Random mutation randomly modifies some of the bits of an individual. Local search mutation performs a local search algorithm to help the genetic algorithm to fine tune a good solution.[2] The local search algorithm is a *Reduced Nearest Neighbor* (RNN) (Gates 1972) algorithm considering only the patterns in the stratum assigned to the subpopulation. This algorithm is fast and greatly improves the performance of CCIS.

The two populations evolve cooperatively. Each generation of the whole system consists of N generations of the combination population followed by M generations of the selector population. The relationship between the two populations can be seen in Fig. 2. The figure shows the basic scheme of the architecture of our cooperative model. First, we divide the whole dataset in as many disjoint equal sized subsets as subpopulations of selectors. Each subset is assigned to a subpopulation, which disregards the rest of the instances. Each subpopulation evolves considering only this subset as the problem to solve. In this way, if we have n training instances and s subpopulations, each subpopulation deals with n/s instances. A member of a subpopulation is a selection of instances in the subset assigned to the subpopulation and is represented by means of a string of 1's and 0's. At a higher level, to accomplish the global view we need to solve the instance selection problem efficiently, we have the population of combinations. Each individual of this population is a selection in the complete dataset. This selection is formed taking a member of every subpopulation. Thus, an individual of this population is made up of an individual of subpopulation 1, an individual of subpopulation 2, and so on. The individuals of the population of combinations are represented as a string of integers, each one indicating the selector from the corresponding subpopulation that belongs to the individual.

The second basis of our model is the use of multiple criteria in the evaluation of the fitness of the individuals of the population of selectors and combinations. The evaluation of

[2]The use of a local search algorithm is common in evolutionary computation as these kinds of algorithms usually find it difficult to converge to an optimal solution.

several objectives for each selector allows the model to encourage cooperation, rewarding the selectors not only for their performance in solving the given problem, but also for other aspects, such as whether they are different from other selectors, whether they are useful in the combinations or anything else considered relevant by the designer.

Each individual in the population of selectors is evaluated combining three different criteria. Each criterion is intended to evaluate the individual in a different aspect. We must note that the fitness of the selectors is not directly measurable within the problem framework as they constitute only a partial solution. These three criteria are:

Error. (ϵ) Training accuracy when applying a 1-NN learning rule[3] using the instances selected by the individual, and only considering the instances in the subset assigned to the subpopulation. It is only an estimation of the ability of the individual as the instances of the other subpopulations are not considered.

Reduction. (ρ) Percentage of reduction represented by the individual. It is measured as the number of instances of the individual, that is, the number of bits set to 1, divided by the size of the instance subset.

Difference. (δ) The individual is removed from all the combinations where it is present, and the performance of such combinations with the individual removed is measured. The value of this criterion is measured as the difference in performance of these combinations with and without the individual. This criterion enforces competition among subpopulations of selectors preventing subpopulations with few useful instances to harm the storage reduction of the algorithm. If a subpopulation does not contain useful instances, the value of this criterion in the fitness of their individuals will be near 0 and only a few instances will be included in the final solution.

The fitness of the individuals is measured using a weighted sum of these three criteria. In this way, the fitness of individual i, f_i, is given by:

$$f_i = w_\epsilon(1-\epsilon) + w_\rho\rho + w_\delta\delta, \tag{1}$$

where w_ϵ, w_ρ, and w_δ, must be fixed by the user. We chose these weights with the constraint $w_\epsilon + w_\rho + w_\delta = 1$.

The evaluation of the combinations is made considering two criteria: reduction of storage, ρ, and classification error, ϵ, using a 1-NN learning rule. The fitness of individual j, F_j, is given by:

$$F_j = w(1-\epsilon) + (1-w)\rho, \tag{2}$$

where $0 \leq w \leq 1$. The weight w is needed to avoid an undesirable effect that may occur due to the asymmetry of the two values of the fitness function: the reduction value can be made arbitrarily high, until the maximum value 1, by just removing more instances. To avoid this effect w must be large, so that if the reduction is too large, the accuracy will be negatively affected and the fitness of the individual penalized.

This fitness function for the combinations is the usual one when evolutionary computation is applied to instance selection (Cano et al. 2003). As we are interested in reducing instances while keeping the performance, the fitness of each individual must take both factors into account. Regarding the fitness function for the population of selectors, we first

[3]By modifying the classifier used in this criterion we can use our model in any learning environment, not just for a k-NN learning rule.

considered a larger set of 14 terms in the fitness function.[4] By means of a stepwise process we removed useless terms until reaching the fitness function given in the equation. The underlying idea is combining the two usual terms in evolutionary instance selection, accuracy and reduction, with a third term that measures the ability of the individual for improving the fitness of the combinations where it participates.

The generation of a new population of combinations is made using a steady-state genetic algorithm (Whitley and Kauth 1988; Whitley 1989).[5] This algorithm is chosen due to the fact that we need a population of combinations that evolves more slowly than the population of selectors, as the changes in the population of combinations have a major impact on the fitness of the selectors. The steady-state genetic algorithm avoids the negative effect that this drastic modification of the population of combinations may have over the subpopulations of selectors. It has also been shown by some papers in the area (Whitley and Starkweather 1990; Syswerda and Study 1991) that the steady-state genetic algorithm produces better solutions than the standard genetic algorithm.

The steady-state algorithm has several features that are different from the standard genetic algorithm. From our point of view, the most interesting is the fact that in each generation only one new individual is created by crossover, and it substitutes the worst individual of the population.

The algorithm also allows adding mutation to the model, always at very low rates. Usually mutation rate ranges from 1% to 5%. In our model we have modified this standard algorithm allowing the replacement of the n worst individuals instead of replacing just the worst one. In our experiments $n = 2$. We also selected the two offspring of the crossover operator, instead of just one.

Crossover is made at selector level, using a standard two-point crossover. Thus the parents exchange their selectors to generate their offspring. Mutation is also carried out at selector level. When a combination is mutated, one of its selectors is randomly chosen and substituted by another one of the same subpopulation selected by means of a roulette algorithm.

During the generation of the new selector subpopulation, some selectors of every subpopulation are removed and substituted by new ones. The removed selectors are also substituted in the combinations. This substitution has two advantages: first, poor performing selectors are removed from the combinations and substituted by potentially better ones; second, new selectors have the opportunity to participate in the combinations immediately after their creation.

The complete algorithm is detailed in Algorithm 1. Mutation in the population of combinations is performed with probability P_{mutation}. In the subpopulations of instance sets random mutation is performed with probability P_{random}. Once a selector is chosen for mutation, each bit is flipped with a probability P_{bit}. RNN mutation is performed with probability P_{rnn}.

The key aspect of our work is the use of two separate populations that cooperate in solving the instance selection problem. This architecture has the following advantages over other approaches:

– The cooperative method approaches the problem of instance selection by means of an automatic decomposition in simpler problems. The cooperation among the individuals that solve those simple problems provides a global solution to the problem.

[4] Many of these terms are common to the ones used for evolving artificial neural networks in a previous work (García-Pedrajas and Ortiz-Boyer 2007).

[5] It is important to note that the specific evolution of both populations is not relevant to the success of the model. It is the cooperative coevolution of the two populations that is the key aspect of the method.

Algorithm 1: Cooperative coevolutionary instance selection (CCIS)

Data: A training set $S = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\}$, a number of nearest neighbors k, and the number of iterations N, M.
Result: The reduced training set: $S' \subset S$.
1 Initialize population of combinations
2 Initialize subpopulations of selectors
while *Stopping criterion not met* **do**
  for N *iterations* **do**
3 Evaluate population of combinations
4 Select two individuals by roulette selection and perform two point crossover
5 Offspring substitutes two worst individuals
6 Perform mutation with probability P_{mutation}
  end
  for M *iterations* **do**
    foreach *subpopulation* i **do**
7 Evaluate selectors of subpopulations i
8 Copy Elitism% to new subpopulation i
9 Fill (1-Elitism)% of subpopulation i by HUX crossover
10 Apply random mutation with probability P_{random}
11 Apply RNN mutation with probability P_{rnn}
    end
12 Evaluate population of combinations
  end
end

- Unlike other approaches that are specifically designed for k-NN classifiers, our algorithm can be used with any classifier. In this way, the model can be used for any instance based classifier, such as a neural network, a tree or a support vector machine.
- The flexibility of the weight of classification accuracy and storage requirements in the fitness of the individuals provides a way to adjust the algorithm to the special needs of any task. Therefore, if we are interested either in a large storage reduction or in a good classification accuracy we can adjust the weights accordingly.
- The proposed method can be distributed with a few modifications. Then next section shows how the distribution can be carried out. Section 5.2 shows the performance of the decoupled version of the algorithm that can be implemented in a distributed/parallel architecture.

One important aspect of any instance selection algorithm is its computational cost. Our algorithm is competitive in this matter with standard genetic algorithms such as CHC. In fact its computational cost is smaller for medium to large problems. In order to study the cost of the algorithm, we must consider the two inner loops of Algorithm 1 separately. The first one, lines 3 to 6, is similar in cost to a CHC algorithm as evaluating an individual will spend the same amount of time. However, as this loop uses a steady-state genetic algorithm, only 2 individuals need to be evaluated per loop, which is a significantly smaller quantity than the evaluation of the whole population needed by the CHC algorithm. This is precisely one of the advantages of our method, as it does not need to evaluate a large number of individuals, taking advantage of the evolution of the subpopulations of selectors.

The second loop consists of the evolution of each subpopulation of selectors. Due to the distribution of the instances among the subpopulations the evolution of these subpopulations is faster. If we have n instances in a dataset, for evaluating a population of a CHC algorithm we need a number of operations proportional to n^2. In our case, if we have s subpopulations, each one deals with a subset of n/s instances, needing a number of operations proportional

to n^2/s^2. Thus, evaluating all s subpopulations will need a number of operations proportional to n^2/s, which is smaller than the number of operations needed by CHC. In fact, the evaluation can be made in parallel if we have the appropriate architecture, of distributedly with the modification shown in the next section, allowing the parallel evaluation of the subpopulations with a number of operations proportional to n^2/s^2. Furthermore, as the number of instances of each subpopulation is smaller, we can use a cache of distances, which can greatly improve the execution time of the algorithm. As the final step of this loop, we must reevaluate the population of combinations.

In principle, this evaluation is as costly as a generation of a CHC algorithm, with the addition of the computation of difference criterion. However, there are several factors that make it simpler. First, the neighbors obtained in the evaluation of the subspopulations can be used in the evaluation of the individuals, decreasing the number of distances to calculate. Also, as the convergence of the subpopulations is faster, the number of selected instances is lower. And, finally, as we use elitism in the subpopulations of selectors, some of the individuals of the population of combinations keep all their selectors unchanged, making their reevaluation unnecessary. Furthermore, the difference criterion can be evaluated efficiently as only the instances which have as nearest neighbor one of the elements of the selector must be reevaluated.

The last step that is responsible for a significant computational cost is the mutation operation. For both, CCIS and CHC, we have used a RNN mutation as local search operator. As it will be shown in Sect. 5.1, this mutation is important for a better performance of the algorithms. However, the computational cost of this mutation is different depending on the method. For CCIS, as the number of instances in each subpopulation is n/s the number of operations for each mutation is proportional to n^2/s^2; for CHC the number of operations is proportional to n^2.

3.1 Distributed implementation

One of the most straightforward ways of scaling up an algorithm is running it in parallel. Depending on the available resources, we can consider the parallel execution of the algorithm in a multiprocessor machine with shared memory (we will call this option *parallel implementation*), or we can run the algorithm using a cluster of independent computers which must communicate with each other (we will call this *distributed implementation*). The parallel implementation has the serious drawback that multiprocessor machines are very expensive. On the other hand, clusters of personal computers are easily available due to their lower cost.

Thus, one major problem of our proposed algorithm is that its implementation in a distributed architecture is problematic due to the coupled evolution of combinations and selectors. Although the evaluation of selectors of different subpopulations can be performed separately, the need to evaluate all the combinations after each modification of the subpopulations (line 12 in Algorithm 1) prevents any efficient distributed implementation of the algorithm. For a distributed implementation we will need to broadcast all the new subpopulations to allow the evaluation of the population of combinations. After that evaluation the new fitness vector will also be needed by all the subpopulations. In this way, the amount of communication would prevent an efficient execution. Although this can be ameliorated by the use of a parallel implementation, this will only be possible when a parallel architecture is available. In contrast, the version we present in this section can be efficiently distributed. As the requirements for a distributed implementation are more readily available, this version of the algorithm is better for dealing with large problems.

Algorithm 2: Decoupled cooperative coevolutionary instance selection (D–CCIS)

Data: A training set $S = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\}$, a number of nearest neighbors k, and the number of iterations N, M.
Result: The reduced training set: $S' \subset S$.
1 Initialize population of combinations
2 Initialize subpopulations of selectors
while *Stopping criterion not met* **do**
  for N *iterations* **do**
3 Evaluate population of combinations
4 Select two individuals by roulette selection and perform two point crossover
5 Offspring substitutes two worst individuals
6 Perform mutation with probability P_{mutation}
  end
  for M *iterations* **do**
    foreach *subpopulation* i **do**
7 Evaluate selectors of subpopulations i
8 Copy Elitism% to new subpopulation i
9 Fill (1-Elitism)% of subpopulation i by HUX crossover
10 Apply random mutation with probability P_{random}
11 Apply RNN mutation with probability P_{rnn}
    end
  end
12 Evaluate population of combinations
end

As stated, the main problem with a distributed implementation is the coupling of the evolution of the M generations of the population of combinations and the N generations of the subpopulations of selectors. We propose an alternative implementation where the evaluation of the combinations is made only after the N generations of the subpopulations of selectors. The general evolution is shown in Algorithm 2. We call this implementation the *decoupled* version of the algorithm.

The main drawback of this version is that during the inner loop (lines 7 to 11) the evaluation of the selectors is only approximate, as the population of combinations is not reevaluated until line 12. Nevertheless, Sect. 3.1 presents results that show that this decoupled implementation is able to achieve almost as good a performance as the original one in terms of storage reduction.

This version allows the distribution of the evaluation and evolution of the different subpopulations of selectors. This is the most computationally expensive part of the algorithm. Additionally, the evaluation of the combinations can be made in parallel, without any modification of the algorithm.

3.2 Evaluating instance selection algorithms

The evaluation of a certain instance selection algorithm is not a trivial task. We can distinguish two basic approaches: direct and indirect evaluation (Liu and Motoda 2002). Direct evaluation evaluates a certain algorithm based exclusively on the data. The objective is to measure at which extent the selected instances reflect the information present in the original data. Some proposed measures are entropy (Cover and Thomas 1991), moments (Smith 1998), and histograms (Chaudhuri et al. 1998). Indirect methods evaluate the effect of the instance selection algorithm on the task at hand. So, if we are interested in classification we evaluate the performance of the used classifier when using the reduced set obtained after instance selection as learning set.

Therefore, when evaluating instance selection algorithms for instance learning, the most usual way of evaluation is estimating the performance of the algorithms for a set of benchmark problems. For those problems several criteria can be considered, such as Wilson and Martinez (2000): storage reduction, testing accuracy, noise tolerance, and learning speed. Speed considerations are difficult to measure, as we are evaluating not only an algorithm but also a certain implementation, so we do not consider speed in our evaluation. We will test our algorithm and other eight algorithms on several datasets and evaluate their data reduction ability and testing accuracy. Section 5.3 is concerned with the study of the noise tolerance of the algorithms.

4 Experimental setup

In order to make a fair comparison between the standard algorithms and our proposal we have selected a large set of 50 problems from the UCI Machine Learning Repository (Hettich et al. 1998). For estimating the storage reduction and testing error we used k-fold cross-validation method, with $k = 10$. In this method the available data is divided into k approximately equal subsets. Then, the method is learned k times, using in turn each one of the k subsets as testing set, and the remaining $k - 1$ subsets as training set. The estimated error is the average testing error of the k subsets. A summary of these datasets is shown in Table 1. The table shows the testing error of a 1-NN classifier, that can be considered as a baseline measure of the error of each dataset. These datasets can be considered representative of problems from small to medium size.

The use of t-tests (Anderson 1984) for the comparison of several classification methods has been criticized in several papers (Dietterich 1998). This test can provide an accurate evaluation of the probability of obtaining the observed outcomes by chance, but it has limited ability to predict relative performance even on further data set samples from the same domain, let alone on other domains. Moreover, as more datasets and classification algorithms are used, the probability of type I error, a true null hypothesis incorrectly rejected, increases dramatically. Multiple comparison tests can be used in order to circumvent this last problem, but these tests are usually not able to establish differences among the algorithms.

To avoid these problems we perform, in a first approach, a single significance test for every pair of algorithms. This test is a sign test on the win/draw/loss record of the two algorithms across all datasets. If the probability of obtaining the observed results by chance, the p-value of the sign test, is below 5% we conclude that the observed performance is indicative of a general underlying advantage to one of the algorithms with respect to the type of learning task used in the experiments.

Nevertheless, the comparison using sign tests has two potential problems: Firstly, the differences between the two algorithms compared must be very marked for the test to find significant differences (Demšar 2006); secondly, on some occasions the p-value of the test can be above or below the critical value due to a single modification of the outcome of one experiment, making the result of the test less reliable. So, as an additional test we have used the Wilcoxon test for comparing pairs of algorithms, for several reasons (Demšar 2006). Wilcoxon test assumes limited commensurability. It is safer than parametric tests since it does not assume normal distributions or homogeneity of variance. Thus, it can be applied to error ratios and storage requirements. Furthermore, empirical results (Demšar 2006) show that it is also stronger than other tests.

Our model is tested against three of the most successful state-of-the-art algorithms. We have used the algorithms IB3 (Aha et al. 1991), DROP3 (Wilson and Martinez 2000), and

Table 1 Summary of datasets. The features of each data set can be C (continuous), B (binary) or N (nominal). The Inputs column shows the number of inputs, as it depends not only on the number of input variables but also on their type

Data set	Cases	Features			Classes	Inputs	1-NN error
		C	B	N			
abalone	4177	7	–	1	29	10	0.8034
anneal	898	6	14	18	5	59	0.0157
audiology	226	–	61	8	24	93	0.3273
autos	205	15	4	6	6	72	0.3300
balance	625	4	–	–	3	4	0.2226
breast-cancer	286	–	3	6	2	15	0.3714
cancer	699	9	–	–	2	9	0.0479
card	690	6	4	5	2	51	0.2174
dermatology	366	1	1	32	6	34	0.0472
ecoli	336	7	–	–	8	7	0.2060
gene	3175	–	–	60	3	120	0.2647
german	1000	6	3	11	2	61	0.3120
glass	214	9	–	–	6	9	0.2952
glass-g2	163	9	–	–	2	9	0.2000
heart	270	13	–	–	2	13	0.2333
heart-c	302	6	3	4	2	22	0.2400
hepatitis	155	6	13	–	2	19	0.1933
horse	364	7	2	13	3	58	0.3667
hypothyroid	3772	7	20	2	4	29	0.0692
ionosphere	351	33	1	–	2	34	0.1314
iris	150	4	–	–	3	4	0.0467
kr vs. kp	3196	–	34	2	2	38	0.0828
labor	57	8	3	5	2	29	0.0600
led24	200	–	24	–	10	24	0.5350
liver	345	6	–	–	2	6	0.3794

ICF (Brighton and Mellish 2002). In IB3 algorithm, an instance is *acceptable* for the algorithm if the lower bound on its accuracy is significantly higher than the upper bound on the frequency of its class at a 90% confidence level (see Wilson and Martinez 2000 for details). On the other hand, an instance is removed if the upper bound of its accuracy is lower (at a 70% confidence level) than the lower bound on the frequency of its class. Other instances are kept during training, and then dropped at the end if they do not prove to be acceptable. The formula for the upper and lower bounds of the confidence interval is:

$$\frac{p + z^2/2n \pm z\sqrt{\frac{p(1-p)}{n} + \frac{z^2}{4n^2}}}{1 + z^2/n}, \tag{3}$$

where for the accuracy of an instance, n is the number of attempts since the introduction of the instance, p is the accuracy of the attempts, and z is the confidence level (0.9 for acceptance and 0.7 for dropping). For the frequency of a class, p is the frequency, n is the number of processed instances, and z the confidence level (0.9 for acceptance and 0.7 for

Table 1 (*Continued*)

Data set	Cases	Features			Classes	Inputs	1-NN error
		C	B	N			
lrs	531	101	–	–	10	101	0.1887
lymphography	148	3	9	6	4	38	0.1929
new-thyroid	215	5	–	–	3	5	0.0333
optdigits	5620	64	–	–	10	64	0.0256
page-blocks	5473	10	–	–	5	10	0.0362
pendigits	10992	16	–	–	10	16	0.0066
phoneme	5404	5	–	–	2	5	0.0952
pima	768	8	–	–	2	8	0.3013
post-operative	90	1	–	7	3	20	0.4889
primary-tumor	339	–	14	3	22	23	0.6515
promoters	106	–	–	57	2	114	0.2500
satimage	6435	36	–	–	6	36	0.0927
segment	2310	19	–	–	7	19	0.0355
sick	3772	7	20	2	2	33	0.0430
sonar	208	60	–	–	2	60	0.1550
soybean	683	–	16	19	19	82	0.0779
texture	5500	40	–	–	11	40	0.0105
tic-tac-toe	958	–	–	9	2	9	0.0779
vehicle	846	18	–	–	4	18	0.2929
vote	435	–	16	–	2	16	0.0675
vowel	990	10	–	–	11	10	0.2919
waveform	5000	40	–	–	3	40	0.2860
wine	178	13	–	–	3	13	0.0353
yeast	1484	8	–	–	10	8	0.4689
zoo	101	1	15	–	7	16	0.0600

dropping). IB3 is one of the *classic* instance selection algorithms, and, as shown by the experiments below, has a very good performance.

DROP3 (*Decremental Reduction Optimization Procedure 3*) represents one of the examples of a new generation of algorithms that were designed taking into account the effect of the order of removal on the performance of the algorithm. So, this algorithm is designed to be insensitive to the order of presentation of the instances. It includes a noise filtering step using a method similar to Wilson's *Edited Nearest-Neighbor* Rule (Wilson 1972). Then, the instances are ordered by the distance to their nearest neighbor. The instances are removed beginning with the instances furthest from its nearest neighbor. This tends to remove the instances furthest from the boundaries first.

For ICF algorithm *coverage* and *reachability* are defined as follows:

$$Coverage(c) = \{c' \in T : LocalSet(c)\} \tag{4}$$

$$Reachable(c) = \{c' \in T : LocalSet(c')\}. \tag{5}$$

The Local-set of a case c is defined as "the set of cases contained in the largest hypersphere centered on c such that only cases in the same class as c are contained in the hyper-

sphere" (Brighton and Mellish 2002). In Case Base Reasoning (CBR) framework (Smyth and Keane 1995) a case c can be adapted to a case c' if c is relevant to the correct prediction of c'. That means that c is a member of the neighborhood of c', bounding the neighborhood of c' by the first instance of a different class (see Brighton and Mellish 2002 for details). The algorithm is based on repeatedly applying a deleting rule to the set of retained instances until no more instances fulfill the deleting rule.

The concept of reachable and coverage sets used by ICF are similar to the neighborhood and associate sets used by RT algorithms (Wilson and Martinez 1997). The difference is that the sets defined in ICF are not of fixed size, but bounded by the first instance belonging to another class. This difference is considered *crucial* by the authors of ICF.

As our method is an evolutionary algorithm, comparing with other evolutionary approaches is a must. We have chosen as evolutionary method to compare with, a genetic algorithm using CHC methodology (Eshelman 1990). The CHC algorithm was used because in Cano et al. (2003) several evolutionary algorithms were compared for instance selection purposes and CHC was found to be the best alternative. CHC stands for *Cross generational elitist selection, Heterogeneous recombination and Cataclysmic mutation*. The non-traditional CHC genetic algorithm differs from traditional GAs in a number of ways (Louis and Li 1997):

1. To obtain the next generation for a population of size N, the parents and the offspring are put together and the N best individual are selected.
2. To avoid premature convergence, only different individuals, separated by a threshold Hamming distance—in our implementation 4 bits—are allowed to mate.
3. During crossover, two parents exchange exactly half of their non-matching bits. This operator is the *Half Uniform Crossover* (HUX) (Eshelman 1990) explained above.
4. Mutation is not used during the regular evolution. In order to avoid premature convergence or stagnation of the search, the population is reinitialized when the individuals are not diverse. In such a case only the best individual is kept in the new population.

The implementation of the genetic algorithm uses the most natural representation for each individual. The chromosome of each individual has as many bits as instances in the training set. A bit whose value is 1 means that the corresponding instance is selected, a 0 means that the corresponding instance is not selected. The fitness measure of an individual i, f_i is given by:

$$f_i = w(1 - \epsilon_i) + (1 - w)\rho_i, \tag{6}$$

where $w = \frac{2}{3}$. Obviously, the objective of the genetic algorithm is to maximize the accuracy and to minimize the storage requirements. The relative weights of storage and accuracy are selected to avoid a large reduction with the side effect of a poor performance.

All the standard methods are based on distances and nearest neighbor classifiers, so our cooperative algorithm uses 1-NN as classification rule to make the comparison as fair as possible. The source code in C of the standard algorithms and CCIS is licensed under the General Public License and freely available upon request to the authors.

5 Experimental results

In this section we show the results obtained with the different tested methods. Table 2 shows the results in storage and testing error terms of the three standard algorithms and a genetic

Table 2 Testing error and storage requirements for the three standard algorithms, IB3, DROP3, and ICF, and the CHC genetic algorithm

Dataset	IB3		DROP3		ICF		Gen. alg. (CHC)	
	Storage	Error	Storage	Error	Storage	Error	Storage	Error
abalone	0.7438	0.8038	0.2450	0.7775	0.2818	0.7854	0.4483	0.8002
anneal	0.0969	0.0427	0.1265	0.0427	0.2197	0.0528	0.0745	0.0427
audiology	0.4706	0.3727	0.2745	0.5500	0.3245	0.5000	0.1784	0.4455
autos	0.4195	0.3750	0.4054	0.4700	0.4335	0.4500	0.1703	0.4400
balance	0.2426	0.2790	0.2179	0.1694	0.1862	0.2258	0.1277	0.2194
breast-cancer	0.1225	0.4214	0.1814	0.3036	0.2190	0.3250	0.0822	0.3607
cancer	0.0310	0.0464	0.0638	0.0507	0.0375	0.0464	0.0549	0.0421
card	0.1435	0.2594	0.2532	0.2319	0.2409	0.2087	0.0837	0.2667
dermatology	0.1294	0.0889	0.1385	0.0667	0.1982	0.0694	0.0979	0.0861
ecoli	0.2310	0.2363	0.1620	0.1636	0.1360	0.2000	0.1327	0.1909
gene	0.2874	0.3117	0.3770	0.2940	0.3265	0.3145	0.4320	0.2991
german	0.1461	0.3870	0.2559	0.2910	0.1880	0.3090	0.1447	0.3230
glass	0.3384	0.3381	0.2922	0.3238	0.2741	0.3333	0.1529	0.3762
glass-g2	0.1674	0.2313	0.3217	0.2313	0.2327	0.2438	0.1333	0.3000
heart	0.1214	0.2185	0.2111	0.2222	0.1745	0.2370	0.0802	0.2630
heart-c	0.1254	0.2467	0.2063	0.1800	0.2000	0.1867	0.1342	0.2567
hepatitis	0.1029	0.2667	0.1479	0.2000	0.1379	0.2133	0.0814	0.2067
horse	0.2271	0.4528	0.2128	0.3611	0.2235	0.3944	0.0875	0.3722
hypothyroid	0.0765	0.2430	0.0251	0.1064	0.0423	0.0793	0.2166	0.0804
ionosphere	0.1399	0.1086	0.0930	0.1457	0.0519	0.1514	0.1408	0.1371
iris	0.1385	0.0867	0.1652	0.0600	0.3526	0.0800	0.0859	0.1067
kr vs. kp	0.1408	0.1135	0.2377	0.0981	0.3185	0.0611	0.2074	0.1097
labor	0.1846	0.2200	0.3692	0.1600	0.1962	0.1800	0.1154	0.2400
led24	0.6178	0.5450	0.4417	0.5100	0.4356	0.5250	0.2106	0.6050
liver	0.1640	0.4265	0.4129	0.3971	0.3129	0.4265	0.0939	0.4471
lrs	0.2259	0.2132	0.1393	0.1924	0.1178	0.2038	0.1789	0.1868
lymph.	0.1911	0.2357	0.3313	0.2500	0.2575	0.2071	0.1679	0.2714
new-thyroid	0.1077	0.0619	0.1273	0.0619	0.1057	0.0619	0.1670	0.0571
optdigits	0.0817	0.0651	0.1003	0.0514	0.0797	0.0792	0.4419	0.0342
page-blocks	0.0368	0.0761	0.0438	0.0411	0.0448	0.0415	0.2260	0.0430
pendigits	0.0346	0.0219	0.0532	0.0160	0.0520	0.0196	0.4911	0.0089
phoneme	0.0748	0.1680	0.1809	0.1282	0.1777	0.1346	0.2738	0.1398

algorithm using CHC methodology. Table 3 shows the results of the cooperative algorithm for 3, 5, and 10 subpopulations.

Figure 3 shows a plot summary of the results in Table 2, and Fig. 4 shows a plot summary of the results in Table 3 together with the decoupled implementation of the next section. The plots show in the x-axis the storage requirements and in the y-axis the testing error. Better results are shown as points closer to the origin. In it, we can easily see that in terms of testing error all methods achieved similar results, but that in terms of storage reduction, the proposed cooperative method is significantly better than the other tested methods.

Table 2 (*Continued*)

Dataset	IB3		DROP3		ICF		Gen. alg. (CHC)	
	Storage	Error	Storage	Error	Storage	Error	Storage	Error
pima	0.1280	0.3355	0.2225	0.2829	0.1942	0.2842	0.1292	0.3013
post-op	0.1086	0.5333	0.1210	0.3333	0.2543	0.4222	0.2148	0.4000
primary-t	0.6562	0.6819	0.2631	0.5940	0.3428	0.6213	0.1925	0.6879
promoters	0.1896	0.2900	0.3958	0.2800	0.3833	0.3100	0.1813	0.2300
satimage	0.1429	0.1287	0.1263	0.1129	0.0973	0.1330	0.4111	0.1098
segment	0.1003	0.0697	0.1352	0.0641	0.1839	0.0745	0.0951	0.0762
sick	0.0387	0.1369	0.0517	0.0536	0.0627	0.0610	0.2043	0.0525
sonar	0.1824	0.2150	0.3330	0.2750	0.2527	0.2750	0.1351	0.2300
soybean	0.1677	0.1059	0.1868	0.1118	0.4992	0.0853	0.0943	0.1265
texture	0.0685	0.0347	0.0985	0.0318	0.1393	0.0371	0.4870	0.0217
tic-tac-toe	0.1292	0.0495	0.2684	0.0105	0.5443	0.0179	0.1162	0.0569
vehicle	0.3002	0.3321	0.3192	0.3036	0.2869	0.3107	0.1845	0.3440
vote	0.0770	0.1047	0.0928	0.0907	0.1036	0.1070	0.0735	0.0884
vowel	0.2077	0.3192	0.4259	0.3222	0.5807	0.3050	0.1906	0.3394
waveform	0.2851	0.3208	0.2709	0.2876	0.1838	0.3118	0.4956	0.2990
wine	0.1379	0.0647	0.1702	0.0588	0.1441	0.0470	0.1199	0.0647
yeast	0.4142	0.5189	0.2718	0.4696	0.2474	0.4716	0.1626	0.0900
zoo	0.2110	0.0500	0.2319	0.1000	0.4736	0.0900	0.1626	0.0900
Average	0.1981	0.2451	0.2160	0.2186	0.2311	0.2262	0.1873	0.2273

Table 3 Summary of results for the cooperative method in terms of testing error and storage requirements. The results of using 3, 5, and 10 subpopulations are shown

Dataset	3 subpopulations		5 subpopulations		10 subpopulations	
	Storage	Error	Storage	Error	Storage	Error
abalone	0.2795	0.7950	0.4681	0.8055	0.4368	0.8036
anneal	0.1043	0.0528	0.0349	0.0292	0.0271	0.0506
audiology	0.2985	0.4500	0.1902	0.3682	0.1142	0.4045
autos	0.2887	0.3050	0.2108	0.3300	0.1108	0.4350
balance	0.1128	0.2016	0.0346	0.1274	0.0256	0.1306
breast-cancer	0.1120	0.3071	0.0434	0.2786	0.0217	0.2857
cancer	0.0405	0.0334	0.0111	0.0363	0.0138	0.0333
card	0.0703	0.2406	0.0277	0.1739	0.0201	0.1942
dermatology	0.0530	0.0389	0.0600	0.0583	0.0506	0.0583
ecoli	0.0416	0.1454	0.0521	0.1606	0.0455	0.1485
gene	0.1467	0.3363	0.1258	0.3372	0.1565	0.3372
german	0.1268	0.3240	0.0260	0.2560	0.0189	0.2700
glass	0.1870	0.3333	0.1316	0.3333	0.0679	0.3143
glass-g2	0.1258	0.2000	0.0966	0.2063	0.0721	0.2500
heart	0.0584	0.1963	0.0432	0.1741	0.0280	0.2185
heart-c	0.0438	0.1700	0.0312	0.1767	0.0243	0.1667

Table 3 (*Continued*)

Dataset	3 subpopulations		5 subpopulations		10 subpopulations	
	Storage	Error	Storage	Error	Storage	Error
hepatitis	0.0614	0.2400	0.0514	0.2267	0.0250	0.2267
horse	0.1152	0.3667	0.0723	0.3611	0.0320	0.3667
hypothyroid	0.0418	0.0753	0.0607	0.0828	0.0731	0.0851
ionosphere	0.0864	0.1372	0.0471	0.0971	0.0380	0.0857
iris	0.1296	0.0667	0.0511	0.0467	0.0422	0.0733
kr vs. kp	0.2434	0.1182	0.0964	0.1787	0.1431	0.1649
labor	0.1596	0.2200	0.1231	0.1600	0.0789	0.1400
led24	0.2611	0.6250	0.1867	0.5550	0.1033	0.5950
liver	0.1183	0.4147	0.0733	0.3676	0.0476	0.3647
lrs	0.1270	0.1868	0.0573	0.1849	0.0398	0.1717
lymphography	0.1560	0.2286	0.0843	0.2214	0.0619	0.2143
new-thyroid	0.1139	0.0524	0.0402	0.0476	0.0402	0.0476
optdigits	0.1088	0.0512	0.2321	0.0391	0.1602	0.0452
page-blocks	0.0552	0.0472	0.0667	0.0442	0.0713	0.0457
pendigits	0.1058	0.0158	0.2348	0.0123	0.1712	0.0125
phoneme	0.3100	0.1406	0.2920	0.1409	0.2929	0.1426
pima	0.0382	0.2790	0.0276	0.2619	0.0157	0.2579
post-operative	0.0790	0.3000	0.0432	0.3333	0.0284	0.3444
primary-tumor	0.2399	0.6364	0.2530	0.6606	0.3219	0.6758
promoters	0.1229	0.1500	0.1708	0.2400	0.2083	0.3100
satimage	0.1323	0.1186	0.1770	0.1102	0.1906	0.1115
segment	0.1143	0.0939	0.1360	0.0779	0.1399	0.0853
sick	0.0420	0.0517	0.0651	0.0522	0.0724	0.0517
sonar	0.1511	0.1800	0.1165	0.1900	0.0739	0.2750
soybean	0.1867	0.1250	0.0775	0.0838	0.0655	0.0853
texture	0.0896	0.0369	0.1398	0.0343	0.1810	0.0275
tic-tac-toe	0.1337	0.0663	0.0593	0.0642	0.0400	0.0737
vehicle	0.1785	0.3345	0.0768	0.3452	0.0551	0.3298
vote	0.0623	0.0814	0.0219	0.0605	0.0161	0.0605
vowel	0.2328	0.3515	0.1598	0.3757	0.1308	0.3798
waveform	0.3702	0.2924	0.4061	0.2906	0.3634	0.3060
wine	0.0565	0.0353	0.0460	0.0529	0.0379	0.0235
yeast	0.1423	0.4851	0.0371	0.4358	0.0276	0.4007
zoo	0.1110	0.0600	0.1011	0.0700	0.0978	0.0800
Average	0.1353	0.2159	0.1094	0.2071	0.0944	0.2152

Table 4 shows the parameters used for the evolution of CCIS. The method is fairly stable with respect to minor changes in these parameters. In this way, small modifications in the number of individuals per population, the values for mutation rates or the rest of the parameters have little effect on the overall performance of the algorithm. The parameters are common for all the experiments shown throughout the paper, unless changes are reported.

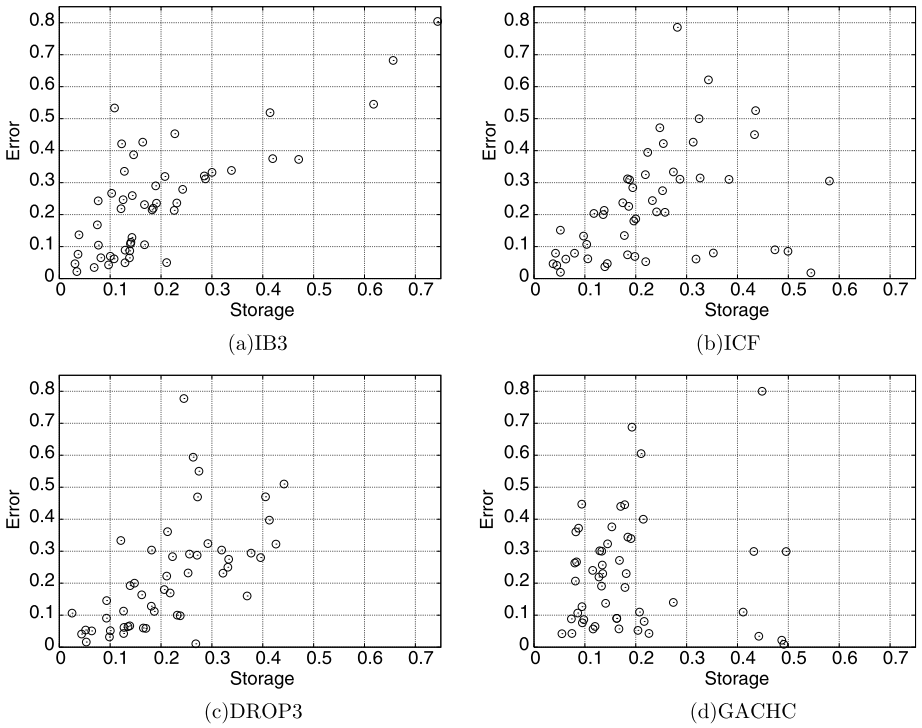
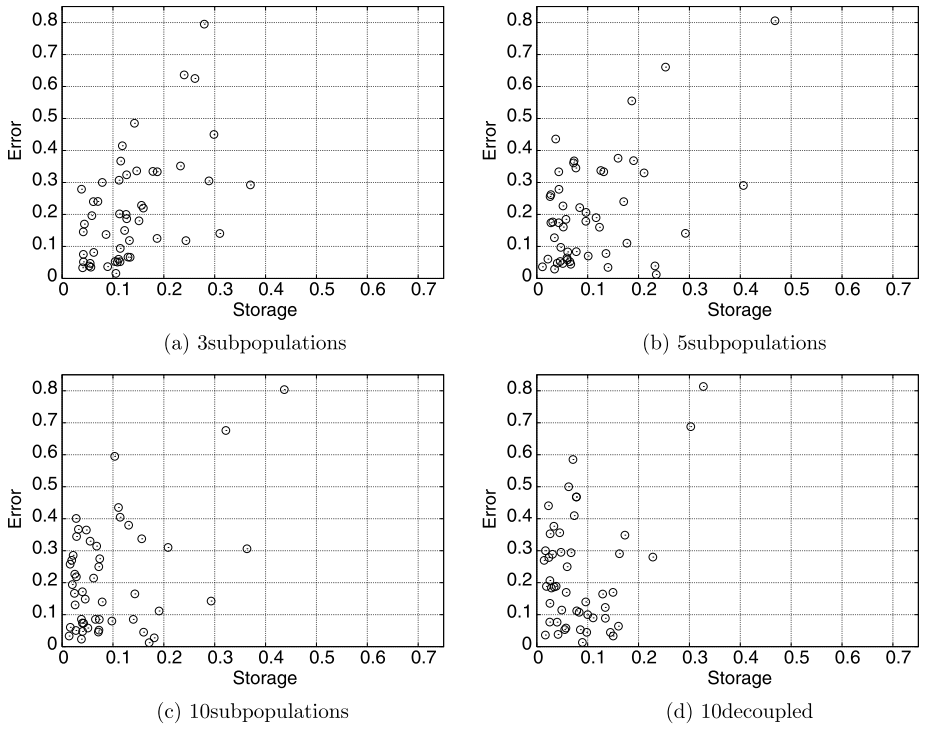


Fig. 3 Summary of results in terms of testing error and storage for the studied methods

The first noticeable result shown by the table is the large improvement on storage requirements achieved by the cooperative algorithm. There is a clear advantage over both the standard algorithms and the CHC algorithm. The improvement is more marked as the number of subpopulations increases. This improvement is achieved without harming the performance, as the testing ability is similar, and better in many problems, to the performance of the other four algorithms. As the cooperative algorithm removes instances taking into account only the effect of the removal on the classification accuracy, it is able to remove instances that are not considered for removal by the standard algorithms. Additionally, dividing the problem into several subproblems that are optimized individually allows a more efficient search for useful solutions. The overall population of combinations assures the collaboration of the best selectors from each subpopulation. The whole system shows a very good performance, both in testing error and storage requirements.

Tables 5 and 6 show the comparison of the different models as explained above in terms of storage requirements and testing error respectively. As we have stated, our first comparative descriptive statistic is the win/draw/loss record. In the table the win/draw/loss record is labeled as s, the first value (the *win* record) is the number of datasets for which the algorithm of the corresponding column (the evolutionary method) performs better than the algorithm of the corresponding row (the standard method), the second value (the *draw* record) is the number of datasets for which the two algorithms have the same error , and the third value (the *loss* record) is the number of datasets for which the algorithm of the corresponding column performs worse than the algorithm of the corresponding row. The row labeled p_s is the



(a) 3subpopulations (b) 5subpopulations

(c) 10subpopulations (d) 10decoupled

Fig. 4 Summary of results in terms of testing error and storage for the studied methods

result of the two-tailed sign test on the win-loss record and the row labeled p_w shows the result of the Wilcoxon test.

The tables also present the mean of errors and storage across all datasets. This is a very gross indication of the relative performance. Although this measure must be handled with caution, a low mean error/storage can be considered indicative of a tendency toward low error rates/storage requirements for individual domains.

Considering the standard algorithms, Table 5 shows that IB3 is able to outperform both ICF and DROP3 in terms of storage requirements. At a confidence level of 95% the sign and Wilcoxon tests show a significant difference between IB3 and the other two standard algorithms. The performance of ICF and DROP3 is very similar, without significant differences between them. Although the CHC algorithm is able to obtain better results than the standard algorithms, the differences are not significant for the Wilcoxon test at a 95% confidence level. The results for the cooperative algorithm are very good; CCIS is able to outperform the three standard algorithms and CHC algorithm. The differences are significant for the two tests, and the win/draw/loss record shows a clear advantage of the proposed method.

For the cooperative method, the increment in the number of subpopulations shows an improvement in the storage requirements. In this way, the algorithm with 10 subpopulations achieves significantly better results than the algorithm with 3 and 5 subpopulations, and the algorithm with 5 subpopulations performs better than the algorithm with 3 subpopulations.

In terms of testing error, Table 6 shows a different behavior. In this case, DROP3 significantly outperforms the other two standard algorithms, and ICF performs better than IB3. CHC achieves results similar to IB3 and ICF, and worse than DROP3. For cooperative algo-

Table 4 Parameters of the cooperative algorithm used in all the experiments

	Parameter	Value
General	Individuals	100
	Subpopulations	{3, 5, 10}
	Generations	100
	M	10
	N	10
Combinations	w	0.9
	Mutation rate	0.1
Selectors	Subpopulation size	30
	w_ϵ	0.25
	w_ρ	0.15
	w_δ	0.60
	Elitism	0.5
	P_{rnn}	0.1
	P_{random}	0.1
	P_{bit}	0.1

Table 5 Comparison of results for the three standard methods, the genetic algorithm, and the three cooperative methods in terms of storage requirements

		IB3	ICF	DROP3	GA(CHC)	Cooperative algorithm		
						3	5	10
Mean all		0.1981	0.2311	0.2160	0.1873	0.1353	0.1094	0.0944
IB3	s		17/0/33	15/0/35	33/0/17	35/0/15	41/0/9	39/0/11
	p_s		0.0328	0.0066	0.0328	0.0066	0.0000	0.0001
	p_w		0.0421	0.0236	0.2184	0.0001	0.0000	0.0000
ICF	s			21/0/29	35/0/15	40/0/10	40/0/10	40/0/10
	p_s			0.3222	0.0066	0.0000	0.0000	0.0000
	p_w			0.9807	0.0578	0.0000	0.0000	0.0000
DROP3	s				35/0/15	40/0/10	39/0/11	38/0/12
	p_s				0.0066	0.0000	0.0001	0.0003
	p_w				0.0718	0.0000	0.0000	0.0000
CHC	s					32/0/18	43/0/7	46/0/4
	p_s					0.0649	0.0000	0.0000
	p_w					0.0169	0.0000	0.0000
CCIS 3	s						36/0/14	37/0/13
	p_s						0.0026	0.0009
	p_w						0.0017	0.0003
CCIS 5	s							37/1/12
	p_s							0.0005
	p_w							0.0004

Table 6 Comparison of results for the three standard methods, the genetic algorithm, and the three cooperative methods in terms of testing error

		IB3	ICF	DROP3	GA(CHC)	Cooperative algorithm		
						3	5	10
Mean all		0.2451	0.2262	0.2186	0.2273	0.2159	0.2071	0.2152
IB3	s		32/3/15	39/1/10	28/1/21	36/1/13	41/0/9	36/0/14
	p_s		0.0186	0.0000	0.3916	0.0014	0.0000	0.0026
	p_w		0.0023	0.0001	0.2711	0.0002	0.0000	0.0018
ICF	s			38/1/11	23/1/26	32/0/18	34/1/15	32/3/15
	p_s			0.0001	0.7754	0.0649	0.0094	0.0186
	p_w			0.0077	0.2078	0.1975	0.0021	0.0129
DROP3	s				17/0/33	23/0/27	32/2/16	30/1/19
	p_s				0.0328	0.6718	0.0293	0.1524
	p_w				0.0047	0.6605	0.0436	0.3491
CHC	s					31/2/17	33/0/17	31/0/19
	p_s					0.0595	0.0328	0.1189
	p_w					0.0065	0.0004	0.0076
CCIS 3	s						29/0/21	28/3/19
	p_s						0.3222	0.2430
	p_w						0.0978	0.2905
CCIS 5	s							17/5/28
	p_s							0.1352
	p_w							0.0940

rithms, only the algorithm with 5 subpopulations is able to outperform the three standard algorithms and CHC. With 3 subpopulations the algorithm performs better than IB3 and CHC, but does not improve the results of ICF and DROP3. With 10 subpopulations the algorithm is able to improve the results of the standard algorithms with the exception of DROP3. There are no significant differences among the cooperative algorithm with 3, 5, and 10 subpopulations at a 95% level of confidence. However, the algorithm with 5 subpopulations is the best one at a confidence level of 90% for the Wilcoxon test.

It is also interesting to compare our proposal with the other genetic algorithm used, CHC, in terms of evolution time. Such comparison is plotted in Fig. 5. The figure shows the difference between the time spent by CHC algorithm on average for each problem and the time spent by our method. A positive value means our algorithm is faster. The figure shows that for small problems the time spent by both algorithms is similar with a small advantage of CHC. For these problems the creation and evaluation of several subpopulations is an overload that is not compensated due to the small number of instances. However, for larger problems our algorithm shows a general advantage over CHC. When the number of subpopulations is 3 the behavior is not better, as each subpopulation receives many instances. For 5 and 10 subpopulations our method is faster for most of the larger problems. The average difference for all problems is favorable to our method for 5 and 10 subpopulations. We must also notice that the method presented in Sect. 3.1 can be easily distributed for a faster execution.

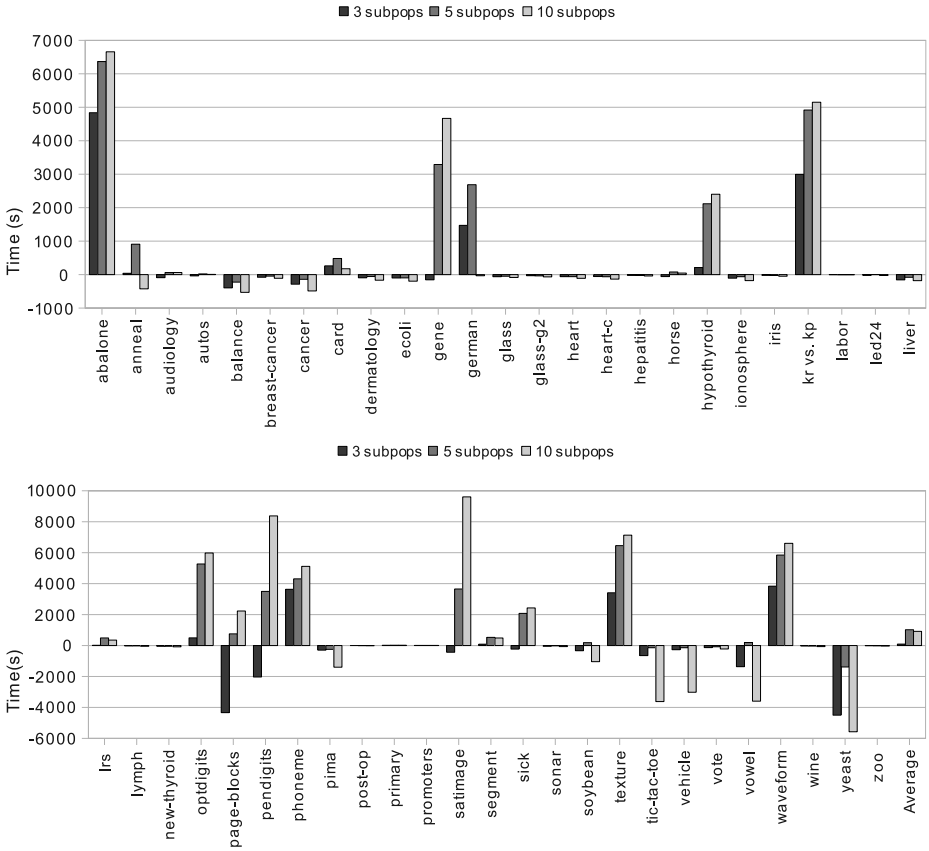


Fig. 5 Summary of average time spent by CCIS with 3, 5, and 10 subpopulations. The table shows the difference in the time needed by CHC and our proposal

5.1 Control experiments

In this section we show several control experiments that test whether some components of our model are responsible for the behavior of the method, or are just adding unneeded complexity to the algorithm. Our model is based on the use of two populations that evolve together. While the population of selectors is obviously a must, the population of combinations can be subject to discussion. For instance, an alternative way would be the combination of the best individual of each subpopulation (Potter and De Jong 2000). We can argue that such architecture is not likely to obtain good results for two reasons: (i) the view that each subpopulation has of the dataset is only partial, so the cooperation among individuals of the different subpopulations introduced by the population of combinations is a must to account for a global view of the problem; and (ii) removing the population of combinations results in also removing the factor "Difference" from the fitness (see (1)) of the selectors. However, we have performed a control experiment using the combination of the best individuals of each subpopulation and removing the population of combinations. The algorithm was performed using 5 subpopulations and the parameters shown in Table 4. The comparison with the proposed method is shown in Table 7. The comparison in terms of testing error is favorable to our proposal with a difference that is statistically significant at a 99% confidence

Table 7 Comparison of results for the cooperative algorithm with 5 subpopulations and the control experiments in terms of testing error and storage requirements

Storage requirements				
		5 subpops	No combinations	No RNN
Mean all		0.1094	0.2971	0.2400
Standard	s		0/0/50	1/0/49
	p_s		0.0000	0.0000
	p_w		0.0000	0.0000
Testing error				
		5 subpops	No combinations	No RNN
Mean all		0.2071	0.2344	0.2116
Standard	s		16/1/33	23/4/23
	p_s		0.0213	1.0000
	p_w		0.0002	0.3903

level. Furthermore, the difference is even more marked in storage requirements, where our method is able to beat the control experiment for all datasets. The poor performance of the control experiment is due to the less degree of cooperation among the subpopulations. In our method, many instances can be removed by the selectors because there are instances in other selectors that are able to make them useless. However, in the control experiment that may not happen, as the collaboration among the subpopulations is no longer explicit, but implicit. An individual will receive a higher fitness if it adds more value to the group of individuals it is being evaluated with from the other populations, producing evolutionary pressure to collaborate. It seems that the explicit collaboration of our method is able to improve the results of that implicit collaboration.

The second control experiment deals with the RNN algorithm performed as a mutation operator. In general, genetic algorithms find problems in fine tuning the solutions obtained after the global search. In the words of Michalewicz (p. 107 in Michalewicz 1994) "Genetic algorithms display inherent difficulties in performing local search for numerical applications. Holland (1975) suggested that the genetic algorithm should be used as a preprocessor to perform the initial search, before turning the search process over to a system that can apply domain knowledge to guide the local search". The same problem is referred in Grefenstette (1987): "Like natural genetic systems, genetic algorithms progress by virtue of changing the distribution of high performance substructures in the overall population; individual structures are not the focus of the attention. Once the high performance regions of the search are identified by a genetic algorithm, it may be useful to invoke a local search routine to optimize the members of the final population". Thus, RNN was introduced as a mutation operator to help the genetic algorithm to locally improve the solutions. Introducing RNN as a mutation, instead of as a local optimization performed in all individuals, was intended to avoid premature convergence of the evolution to local minima. So, although the use of RNN is motivated by the previous facts, we have performed a second control experiment to assure that the algorithm is actually helping the model. The control experiment uses 5 subpopulations with the parameters shown in Table 4, with the exception that RNN mutation probability is set to 0. The comparison is shown in Table 7.

In terms of testing error, both experiments, with and without RNN mutation obtain similar results. As RNN only removes instances that do not cause a decrement in the accuracy, its presence or absence should not affect the testing error of the method. On the other hand,

Table 8 Comparison of results for the cooperative algorithm with the standard and the decoupled implementation with 10 subpopulations in terms of testing error and storage requirements

		Storage		Testing error	
		CCIS	D–CCIS	CCIS	D–CCIS
Mean all		0.0944	0.0841	0.2152	0.2283
Standard	s		25/0/25		14/2/34
	p_s		1.0000		0.0055
	p_w		0.3823		0.0013

the results for storage are greatly affected by removing RNN mutation. For all datasets, except 1, the algorithm performs worse without RNN mutation, showing the utility of such a mutation operator.

5.2 Distributed implementation

In Sect. 3.1 we offered a version of the algorithm that can be distributed. The possibility of implementing these kinds of algorithms in parallel or distributedly is of great importance due to the execution time requirements of the algorithm when facing large problems. In this section we study whether the approximate implementation discussed achieves good enough results to be used when the distributed implementation is a necessity due to the size of the problems.

For testing the performance of the decoupled implementation we have run this version of the algorithm with the same parameters shown in Table 4 and 10 subpopulations. These results are compared with the standard version of the cooperative algorithm. The results are shown in Table 9.

The comparison of the two versions of the algorithm is shown in Table 8. In terms of storage requirements it is interesting to note that the decoupled algorithm is able to match the performance of the standard implementation. The mean over all datasets is even better than the mean of the standard version, although the differences between them are not significant. On the other hand, the standard implementation performs significantly better than the decoupled version in terms of testing error. This is an expected result, as the algorithm is iterating with only an approximate knowledge of the classification error of the individuals. Nevertheless, the differences are not dramatic, and the decoupled algorithm is an interesting choice when the problem to be faced is large. This is a clear advantage over other algorithms that cannot be implemented distributedly.

5.3 Noise tolerance

Real data is often affected by noise, which can occur in many forms and for many reasons (Brighton and Mellish 2002). In this section we focus on the study of the effect on the instance selection algorithm of noise in the form of mislabeled training instances. As most of the standard algorithms implement some steps of noise filtering it is very interesting to know the effectiveness of such noise filtering and how it compares with the cooperative process presented here. We should note that the cooperative algorithm has no explicit treatment of noisy instances, so the only way it can remove them is via the effect that noisy instances have on the fitness function.

To add noise to the class labels we follow the method of Dietterich (2000). To add classification noise at a rate r, we chose a fraction r of the instances and changed their class labels to be incorrect choosing uniformly from the set of incorrect labels. We chose all the

Table 9 Summary of results for the cooperative method with 10 subpopulations using the standard and the decoupled implementations

Dataset	CCIS		D–CCIS	
	Storage	Error	Storage	Error
abalone	0.4368	0.8036	0.3274	0.8134
anneal	0.0271	0.0506	0.1605	0.0640
audiology	0.1142	0.4045	0.0779	0.4682
autos	0.1108	0.4350	0.0627	0.5000
balance	0.0256	0.1306	0.0258	0.1355
breast-cancer	0.0217	0.2857	0.0237	0.2786
cancer	0.0138	0.0333	0.0168	0.0362
card	0.0201	0.1942	0.0188	0.1884
dermatology	0.0506	0.0583	0.0573	0.0583
ecoli	0.0455	0.1485	0.0578	0.1697
gene	0.1565	0.3372	0.1731	0.3486
german	0.0189	0.2700	0.0171	0.3000
glass	0.0679	0.3143	0.0736	0.4095
glass-g2	0.0721	0.2500	0.0673	0.2938
heart	0.0280	0.2185	0.0379	0.1889
heart-c	0.0243	0.1667	0.0287	0.1833
hepatitis	0.0250	0.2267	0.0257	0.2067
horse	0.0320	0.3667	0.0262	0.3528
hypothyroid	0.0731	0.0851	0.1107	0.0899
ionosphere	0.0380	0.0857	0.0491	0.1143
iris	0.0422	0.0733	0.0548	0.0533
kr vs. kp	0.1431	0.1649	0.1294	0.1643
labor	0.0789	0.1400	0.0962	0.1400
led24	0.1033	0.5950	0.0711	0.5850
liver	0.0476	0.3647	0.0335	0.3765
lrs	0.0398	0.1717	0.0341	0.1868
lymphography	0.0619	0.2143	0.0597	0.2500
new-thyroid	0.0402	0.0476	0.0418	0.0381
optdigits	0.1602	0.0452	0.1447	0.0445
page-blocks	0.0713	0.0457	0.0981	0.0448
pendigits 1	0.1712	0.0125	0.0896	0.0132
phoneme	0.2929	0.1426	0.1497	0.1697
pima	0.0157	0.2579	0.0144	0.2697
post-operative	0.0284	0.3444	0.0308	0.2889
primary-tumor	0.3219	0.6758	0.3026	0.6879
promoters	0.2083	0.3100	0.2281	0.2800

datasets and an intermediate rate of noise of 10%. With this level of noise we performed the experiments using the same setup and parameters of the previous sections. The cooperative method was run with 10 subpopulations as it was the best performing configuration. Table 10 shows the results in terms of testing error and storage requirements of all the algorithms.

Table 9 (*Continued*)

Dataset	CCIS		D–CCIS	
	Storage	Error	Storage	Error
satimage	0.1906	0.1115	0.1346	0.1226
segment	0.1399	0.0853	0.1347	0.0887
sick	0.0724	0.0517	0.0856	0.0531
sonar	0.0739	0.2750	0.0474	0.2950
soybean	0.0655	0.0853	0.0785	0.1118
texture	0.1810	0.0275	0.1500	0.0335
tic-tac-toe	0.0400	0.0737	0.0833	0.1074
vehicle	0.0551	0.3298	0.0450	0.3560
vote	0.0161	0.0605	0.0255	0.0768
vowel	0.1308	0.3798	0.0777	0.4677
waveform	0.3634	0.3060	0.1625	0.2906
wine	0.0379	0.0235	0.0398	0.0765
yeast	0.0276	0.4007	0.0229	0.4405
zoo	0.0978	0.0800	0.1000	0.1000
Average	0.0944	0.2152	0.0841	0.2283

Table 10 Testing error and storage requirements are shown for the three standard algorithms, IB3, DROP3, and ICF, the CHC genetic algorithm, and CCIS with 10 subpopulations, for a noise level of 10%

Dataset	IB3		DROP3		ICF		CHC		CCIS 10	
	Stor.	Error	Stor.	Error	Stor.	Error	Stor.	Error	Stor.	Error
abalone	0.8222	0.8350	0.2529	0.7995	0.1839	0.8072	0.4904	0.8554	0.4472	0.8321
anneal	0.2438	0.3326	0.1271	0.1551	0.1314	0.1831	0.0831	0.2056	0.1260	0.2023
audiology	0.5931	0.5046	0.2917	0.6046	0.1397	0.6091	0.2333	0.6046	0.1010	0.5182
autos	0.4860	0.4750	0.3935	0.5150	0.2184	0.5600	0.2189	0.4700	0.1043	0.4550
balance	0.3222	0.4129	0.2236	0.2920	0.1732	0.3145	0.0704	0.2903	0.0245	0.2371
breast-c.	0.2020	0.5107	0.2620	0.3429	0.1818	0.4214	0.1163	0.3964	0.0209	0.3000
cancer	0.0862	0.1957	0.0643	0.1348	0.0435	0.1580	0.0964	0.1565	0.0095	0.1130
card	0.1926	0.3594	0.2811	0.3174	0.1736	0.3116	0.1142	0.3551	0.0205	0.2652
derma.	0.2976	0.2612	0.1467	0.1167	0.0730	0.2889	0.0891	0.1972	0.0403	0.1389
ecoli	0.4184	0.4515	0.1650	0.2727	0.0904	0.3091	0.1165	0.3424	0.0416	0.2757
gene	0.3518	0.4025	0.3842	0.3552	0.2223	0.4104	0.4710	0.4025	0.1738	0.4117
german	0.2079	0.4390	0.3100	0.3590	0.1361	0.4080	0.0826	0.3830	0.0183	0.3350
glass	0.4182	0.4571	0.3145	0.4048	0.1446	0.4810	0.1254	0.4286	0.0829	0.3810
glass-g2	0.2238	0.2938	0.3340	0.2938	0.1531	0.3563	0.1211	0.3563	0.0571	0.3063
heart	0.1786	0.3296	0.2407	0.2889	0.1304	0.3259	0.1054	0.3185	0.0280	0.2296
heart-c	0.2048	0.3533	0.2397	0.2900	0.1552	0.2900	0.0662	0.2700	0.0268	0.2467
hepatitis	0.1393	0.3533	0.1750	0.2067	0.0936	0.2534	0.1079	0.2400	0.0322	0.2400
horse	0.2875	0.5250	0.2226	0.4194	0.1326	0.4167	0.1083	0.4556	0.0277	0.4472
hypo.	0.1882	0.5430	0.0393	0.1833	0.0271	0.2268	0.2764	0.2316	0.0776	0.2456

Table 10 (*Continued*)

Dataset	IB3		DROP3		ICF		CHC		CCIS 10	
	Stor.	Error	Stor.	Error	Stor.	Error	Stor.	Error	Stor.	Error
ionosph.	0.1740	0.3200	0.1592	0.2686	0.0370	0.2829	0.0877	0.2829	0.0376	0.2286
iris	0.2430	0.2667	0.1659	0.1200	0.1222	0.1533	0.0993	0.2133	0.0392	0.1533
kr vs. kp	0.2219	0.2762	0.2514	0.2022	0.2306	0.2204	0.2306	0.2423	0.1371	0.2677
labor	0.2019	0.2600	0.3481	0.2400	0.1269	0.3200	0.1423	0.4000	0.0885	0.3400
led24	0.7084	0.6300	0.4167	0.5550	0.2917	0.6150	0.2128	0.6500	0.0895	0.6350
liver	0.2061	0.4823	0.4357	0.4235	0.2164	0.4324	0.1031	0.4274	0.0428	0.4177
lrs	0.4226	0.4359	0.1555	0.2491	0.0826	0.2717	0.1086	0.3302	0.0370	0.2302
lymph.	0.3433	0.4643	0.3090	0.3214	0.1537	0.3000	0.1515	0.3357	0.0574	0.3357
new-th.	0.1995	0.2190	0.1278	0.0952	0.0706	0.1238	0.1665	0.1619	0.0356	0.0429
optdigits	0.3222	0.3338	0.0982	0.1505	0.0558	0.2080	0.4818	0.2153	0.3237	0.2139
page-bl.	0.2087	0.4731	0.0456	0.1468	0.0268	0.1658	0.3200	0.2108	0.0869	0.2230
pendigits	0.3052	0.3155	0.0540	0.1211	0.0330	0.1468	0.4985	0.1865	0.4067	0.1877
phoneme	0.1533	0.2965	0.1983	0.2267	0.1229	0.2622	0.3142	0.2561	0.3293	0.2624
pima	0.1789	0.3882	0.2552	0.3263	0.1406	0.3487	0.1030	0.3592	0.0173	0.3039
post-op	0.2198	0.6778	0.2605	0.5111	0.2296	0.4889	0.1667	0.3778	0.0197	0.3444
primary-t	0.7448	0.7515	0.2794	0.6667	0.2213	0.6819	0.2206	0.7212	0.3565	0.7394
promoters	0.2146	0.2900	0.4458	0.2700	0.2917	0.2600	0.2250	0.2600	0.2198	0.4100
satimage	0.3258	0.3417	0.1306	0.2040	0.0613	0.2417	0.4962	0.2603	0.1180	0.2628
segment	0.2913	0.2857	0.1348	0.1563	0.0957	0.1861	0.0850	0.2125	0.1362	0.2204
sick	0.1163	0.3769	0.0801	0.1722	0.0541	0.1899	0.3363	0.2180	0.0815	0.2008
sonar	0.1984	0.3100	0.3431	0.3200	0.1362	0.4400	0.0840	0.4150	0.0516	0.3350
soybean	0.3665	0.3941	0.1917	0.2485	0.1903	0.2794	0.1062	0.3191	0.0636	0.2338
texture	0.3193	0.3131	0.0973	0.1302	0.0649	0.1604	0.4231	0.1977	0.3217	0.2029
t-t-t	0.1751	0.2242	0.2564	0.1221	0.2672	0.1547	0.0820	0.1979	0.0348	0.2063
vehicle	0.3788	0.4417	0.3215	0.3714	0.2034	0.4203	0.1031	0.4274	0.0493	0.4072
vote	0.1263	0.2721	0.1281	0.2139	0.0947	0.2442	0.1332	0.2302	0.0189	0.1860
vowel	0.3596	0.4889	0.4112	0.4050	0.2844	0.4374	0.1945	0.5041	0.1223	0.5040
wavef.	0.3563	0.4168	0.2736	0.3488	0.1155	0.3826	0.4487	0.3902	0.3598	0.3930
wine	0.2311	0.2588	0.1721	0.1470	0.0932	0.1529	0.1006	0.2000	0.0367	0.1412
yeast	0.5559	0.6128	0.2942	0.5209	0.1611	0.5473	0.1407	0.3300	0.0252	0.5061
zoo	0.3385	0.2200	0.2341	0.2000	0.3033	0.2000	0.1407	0.3300	0.0879	0.1500
Average	0.3014	0.3975	0.2309	0.2961	0.1436	0.3290	0.1920	0.3365	0.1058	0.3093

Tables 11 and 12 show the comparison of the algorithms in terms of testing error and storage requirements for the experiments with 10% noise level. In terms of storage requirements, ICF is able to improve its results, outperforming DROP3, whose performance deteriorates slightly, and IB3 that is clearly worse. It seems that the noise removing step that is performed by both ICF and DROP3 is useful in noisy problems. The two evolutionary algorithms maintain their results, showing the usual robustness of EC in presence of noise. Although ICF and DROP3 perform well in this noisy environment, CCIS is still able to outperform all algorithms with a difference that is statistically significant.

Table 11 Comparison of results for the three standard methods, the genetic algorithm, and the cooperative method with 10 subpopulations in terms of storage requirements for a noise level of 10%

		IB3	ICF	DROP3	GA(CHC)	CCIS10
Mean all		0.3014	0.1436	0.2309	0.1920	0.1058
IB3	s		45/0/5	31/0/19	36/0/14	44/0/6
	p_s		0.0000	0.1189	0.0026	0.0000
	p_w		0.0000	0.0060	0.0003	0.0000
ICF	s			3/0/47	27/0/23	37/0/13
	p_s			0.0000	0.6718	0.0009
	p_w			0.0000	0.6328	0.0104
DROP3	s				36/0/14	39/0/11
	p_s				0.0026	0.0001
	p_w				0.0591	0.0000
CHC	s					46/0/4
	p_s					0.0000
	p_w					0.0000

Table 12 Comparison of results for the three standard methods, the genetic algorithm, and the cooperative method with 10 subpopulations in terms of testing error for a noise level of 10%

		IB3	ICF	DROP3	GA(CHC)	CCIS10
Mean all		0.3975	0.3290	0.2961	0.3365	0.3093
IB3	s		42/0/8	46/1/3	41/1/8	42/0/8
	p_s		0.0000	0.0000	0.0000	0.0000
	p_w		0.0000	0.0000	0.0000	0.0000
ICF	s			43/2/5	18/2/30	27/1/22
	p_s			0.0000	0.1114	0.5682
	p_w			0.0000	0.0362	0.0384
DROP3	s				7/0/43	21/0/29
	p_s				0.0000	0.3222
	p_w				0.0000	0.0659
CHC	s					33/3/14
	p_s					0.0079
	p_w					0.0002

In terms of testing error, Table 12, all the algorithms suffer from noise in instance labels. However, the effect is here more homogeneous, keeping the differences as in the original case without noise. CHC is greatly affected, and it is now significantly worse that ICF. CCIS, as in the case without noise, is significantly better than IB3, ICF, and CHC, and shows a general performance worse than DROP3 which is statistically significant at a confidence level of 90%.

6 Comparison with recent algorithms

In the previous sections we have compared our method with the most widely used algorithms for instance selection. We can consider the methods used for comparison as "classical" as they have been around for quite a long time and are widely used. In this section we compare our algorithm with more recent methods that are designed following new ideas. In this way, we want to show whether our method is also competitive with modern methods. We have chosen four algorithms:

- Modified Selective Subset (MSS; Barandela et al. 2005) method. This method is a modification of the algorithm of Ritter et al. (1975) for finding a selective subset. A subset is selective if it is consistent and all prototypes in the original training set are nearer to a selective neighbor of the same class than to any member of the training set from a different class. MSS algorithm is aimed at obtaining a minimal consistent subset but using the selective property. In this way the authors define the modified selective subset as that subset of the training set which contains, for every instance x_i in the training set, that element of its neighborhood that is the nearest to a class other than that of x_i. The authors propose an iterative procedure to find this modified selective subset. As in other instance selection algorithms the instances are ordered regarding the distance to its nearest enemy.
- Entropy-based instance selection (Son and Kim 2006). This method is based on a two step procedure. First the original dataset is partitioned using the values of each variable in increasing entropy value. The process is iterated until all the subsets have instances of only one class. Variables that are not used for partitioning the data are not used any more. Then, a second step is performed to obtain the representatives of each subset. The representatives are chosen as the center of each subset together with its k nearest neighbors. As it will be seen in the results, partition using input values is able to obtain fairly good results for nominal data, but very poor results for real-valued inputs.
- Intelligent Multiobjective Evolutionary Algorithm (IMOEA; Chen et al. 2005) method. This method is a multi-objective evolutionary algorithm which considers not only instance selection but also feature selection. The algorithm has three objectives, maximization of training accuracy and minimization of the number of instances and features selected. The multi-objective algorithm used is based on Pareto dominance as it is common in multi-objective algorithms (Zitzler et al. 2003). The fitness of each individual is the difference between the individuals it dominates and the individuals that dominate it. The algorithm also includes a new crossover operator, called *intelligent crossover*, which incorporates the systematic reasoning ability of orthogonal experimental design (Leung and Wang 2001) to estimate the contribution of each gene to the fitness of the individuals.
- LVQPRU method (Li et al. 2005). The method starts choosing randomly N_{pc} prototypes for each class, where N_{pc} is a parameter of the algorithm. Then, the selected prototypes that are not useful for classifying any instance are discarded and an iterative process is started. First a learning vector quantization (LVQ) algorithm is applied to fine-tune the set of selected prototypes, then a standard condensing algorithm (Tomek 1976) is applied. Of the remaining instances the one whose removal causes the least increase in the classification error is discarded. The method continues until we have as many prototypes as classes. The result of the algorithm is the set of prototypes with the minimum error.

In the experiments we use the parameters for each algorithm suggested by the authors. Results are shown in Table 13, and the comparison with our cooperative approach in Table 14. IMOEA and entropy-based instance selection select instances and features, so the

Table 13 Summary of results for the cooperative method in terms of testing error and storage requirements. The results of using 3, 5, and 10 subpopulations are shown

Dataset	MSS		Entropy-based IS		IMOEA		LVQPRU	
	Storage	Error	Storage	Error	Storage	Error	Storage	Error
abalone	0.6435	0.8053	0.9959	0.8036	0.1153	0.8034	0.1105	0.7638
anneal	0.1731	0.0303	0.0806	0.0258	0.0495	0.0101	0.0624	0.0506
audiology	0.4500	0.3864	0.1605	0.4045	0.0732	0.3091	0.5059	0.3091
autos	0.4519	0.3300	0.0883	0.2000	0.0429	0.2600	0.4838	0.3200
balance	0.3169	0.2823	0.9235	0.2065	0.2791	0.2952	0.1197	0.1903
breast-cancer	0.4484	0.4214	0.8496	0.3786	0.0716	0.3143	0.1632	0.3393
cancer	0.1013	0.0783	0.1771	0.0609	0.1004	0.0580	0.0387	0.0377
card	0.3908	0.2652	0.2521	0.2580	0.0432	0.2044	0.0644	0.2014
dermatology	0.1988	0.0917	0.2094	0.0917	0.0535	0.0528	0.5952	0.0361
ecoli	0.3010	0.2515	0.5859	0.2091	0.1265	0.2666	0.3716	0.2121
gene	0.4442	0.3107	0.2071	0.4070	0.0728	0.2027	0.0322	0.1685
german	0.4309	0.3550	0.4530	0.3110	0.0758	0.3240	0.0498	0.3030
glass	0.4207	0.3143	0.6289	0.2714	0.0796	0.4476	0.5539	0.3000
glass-g2	0.3932	0.2500	0.5355	0.2313	0.0642	0.2438	0.2435	0.2438
heart	0.3716	0.2630	0.4946	0.2519	0.0701	0.3111	0.1951	0.2222
heart-c	0.3923	0.2600	0.3461	0.2734	0.0607	0.2967	0.1684	0.2567
hepatitis	0.3186	0.2334	0.3267	0.2933	0.0281	0.2333	0.1607	0.2267
horse	0.4168	0.3917	0.8851	0.3889	0.0599	0.4194	0.2451	0.3528
hypothyroid	0.1675	0.0995	0.1286	0.2687	0.0460	0.0228	0.0092	0.0610
ionosphere	0.2484	0.1457	0.1531	0.1229	0.0406	0.1457	0.1693	0.0971
iris	0.2155	0.0933	0.5217	0.1000	0.0266	0.1667	0.3837	0.0733
kr vs. kp	0.3192	0.0843	0.0420	0.3254	0.1163	0.0354	0.0107	0.0552
labor	0.3327	0.2000	0.1781	0.2600	0.0000	0.6000	0.4096	0.0800
led24	0.5661	0.5900	0.6355	0.5900	0.0584	0.3250	0.8300	0.5750
liver	0.5064	0.3971	0.9249	0.3706	0.0889	0.4147	0.1997	0.3971
lrs	0.2818	0.2132	0.1141	0.2245	0.0607	0.1509	0.2314	0.1736

storage values shown in the tables consider both reductions. In terms of storage reduction, our proposal is significantly better than the three methods based on non-evolutionary procedures, MSS, entropy-based instance selection and LVQPRU. On the other hand, IMOEA, which is also an evolutionary algorithm, is able to match the storage reduction of our method. However, we must bear in mind that IMOEA considers feature selection and our method does not. Regarding testing error, our method with 3 and 5 subpopulations is able to significantly improve the results of all the other methods, with the exception of LVQPRU. With 10 subpopulations is as good as MSS and IMOEA, better than entropy-based instance selection at a confidence level of 90%, and worse than LVQPRU, but with a significantly better storage reduction.

Figure 6 shows a plot summary of the results in Table 13. As in the previous plots, x-axis shows the storage requirements and y-axis the testing error. There, we can easily see that in terms of testing error all methods achieved similar results, but that in terms of storage reduction, the proposed cooperative method is significantly better than the other tested methods, with the exception of IMOEA, which also performs feature selection. The

Table 13 (*Continued*)

Dataset	MSS		Entropy-based IS		IMOEA		LVQPRU	
	Storage	Error	Storage	Error	Storage	Error	Storage	Error
lymphography	0.3948	0.2286	0.1951	0.2500	0.0439	0.2786	0.3396	0.1643
new-thyroid	0.1546	0.0476	0.5292	0.0524	0.0857	0.0571	0.1763	0.0524
optdigits	0.1663	0.0425	0.5020	0.0215	0.1450	0.0374	0.0234	0.0685
page-blocks	0.0991	0.0428	0.3578	0.0417	0.1397	0.0441	0.0170	0.0431
pendigits	0.0900	0.0135	0.6463	0.0088	0.2179	0.0560	0.0107	0.0369
phoneme	0.2433	0.1287	0.7149	0.1072	0.3047	0.1622	0.0073	0.1785
pima	0.4142	0.3342	0.8507	0.3224	0.0544	0.3948	0.0708	0.2855
post-operative	0.4247	0.4556	0.8803	0.4667	0.0370	0.4630	0.2160	0.3889
primary-tumor	0.5056	0.6576	0.8683	0.6758	0.1508	0.6546	0.6628	0.6576
promoters	0.4563	0.3000	0.0464	0.3300	0.0474	0.2600	0.4865	0.2300
satimage	0.2032	0.1212	0.5835	0.1012	0.1592	0.1280	0.0137	0.1030
segment	0.1628	0.0498	0.4615	0.0615	0.0219	0.5282	0.0435	0.0541
sick	0.1240	0.0608	0.1813	0.0886	0.0589	0.0339	0.0016	0.0570
sonar	0.3697	0.1800	0.1496	0.3550	0.0460	0.2050	0.1591	0.2150
soybean	0.2412	0.0897	0.3839	0.1471	0.0836	0.0676	0.1759	0.0911
texture	0.1335	0.0206	0.5331	0.0096	0.1037	0.0249	0.0245	0.0567
tic-tac-toe	0.2092	0.0474	0.4902	0.0327	0.1787	0.2358	0.0306	0.0747
vehicle	0.4139	0.3321	0.6477	0.3012	0.0834	0.3310	0.0906	0.2857
vote	0.1684	0.0930	0.1512	0.1372	0.0503	0.0489	0.0357	0.0930
vowel	0.2971	0.3141	0.8954	0.3030	0.1847	0.4465	0.2263	0.3171
waveform	0.3435	0.3052	0.8755	0.2792	0.0856	0.2516	0.0120	0.1796
wine	0.1801	0.0765	0.3231	0.0823	0.0376	0.1412	0.2429	0.0412
yeast	0.5339	0.5230	0.7232	0.4838	0.1790	0.5500	0.1424	0.4311
zoo	0.1670	0.0700	0.1640	0.0900	0.0349	0.1000	0.4242	0.0500
Average	0.3160	0.2336	0.4610	0.2415	0.0888	0.2483	0.2008	0.2040

plot shows how, in terms of storage MSS, entropy-based instance selection and LVQPRU are clearly worse than CCIS. IMOEA is able to achieve a similar performance, although it is worse in testing error.

7 Large datasets

One of the most interesting aspects in any instance selection algorithm is its scalability. Studying whether it is able to deal with large problems in a reasonable time and achieve a good performance is of relevance, due to the increasing size of the datasets for many of the problems of interest. In this section we study the behavior of the studied algorithms when they face problems that can be considered large. We use the five problems from the UCI Machine Learning Repository shown in Table 15. For krkopt, letter, and magic, we estimate testing error and storage following 10-fold cross-validation as in the previous experiments. For adult and shuttle, due to the large number of instances, we use 5×2 cross-validation (Dietterich 1998). 5×2 cross-validation performs five replications of a two-fold cross-validation.

Table 14 Comparison of results for the three standard methods, the genetic algorithm, and the cooperative method with 3, 5, and 10 subpopulations in terms of storage requirements and testing error

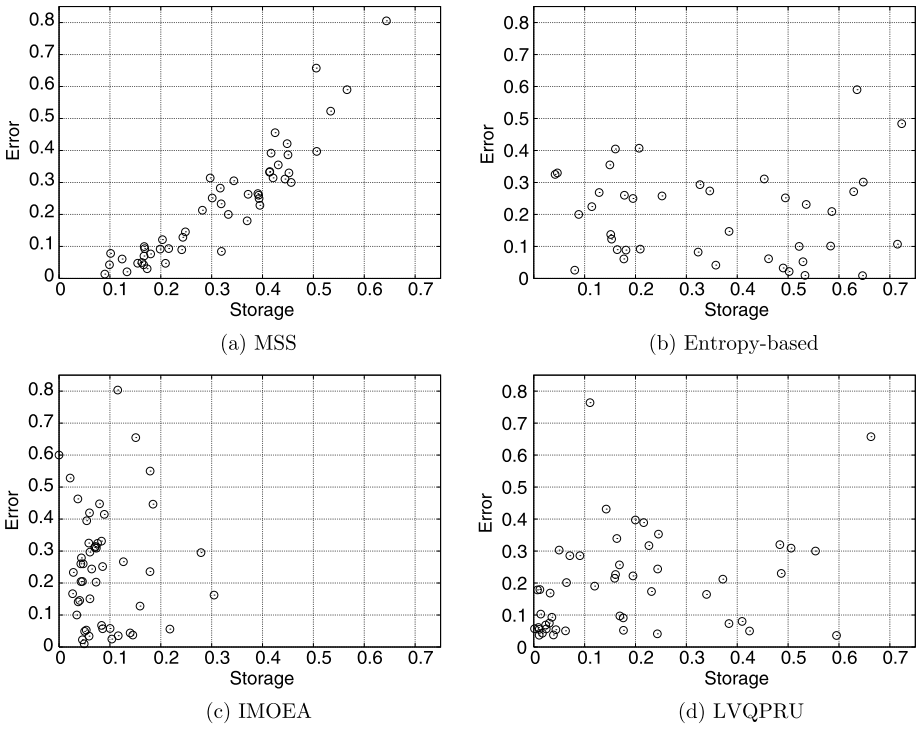
Storage					
		MSS	Entropy-based IS	IMOEA	LVQPRU
Mean all		0.3160	0.4610	0.0888	0.2008
3 subpops	s	5/0/45	4/0/46	27/0/23	14/0/36
	p_s	0.0000	0.0000	0.6718	0.0026
	p_w	0.0000	0.0000	0.2818	0.0035
5 subpops	s	4/0/46	3/0/47	22/0/28	14/0/36
	p_s	0.0000	0.0000	0.4799	0.0026
	p_w	0.0000	0.0000	0.6887	0.0019
10 subpops	s	1/0/49	3/0/47	25/0/25	15/0/35
	p_s	0.0000	0.0000	1.0000	0.0066
	p_w	0.0000	0.0000	0.3823	0.0005
Testing error					
		MSS	Entropy-based IS	IMOEA	LVQPRU
Mean all		0.2336	0.2415	0.2483	0.2040
3 subpops	s	13/3/34	15/0/35	19/0/31	23/1/26
	p_s	0.0031	0.0066	0.1189	0.7754
	p_w	0.0003	0.0002	0.0093	0.7102
5 subpops	s	18/3/29	16/3/31	21/0/29	23/2/25
	p_s	0.1439	0.0400	0.3222	0.8854
	p_w	0.0337	0.0018	0.0959	0.4286
10 subpops	s	22/1/27	20/0/30	24/1/25	29/0/21
	p_s	0.5682	0.2026	1.0000	0.3222
	p_w	0.5054	0.0798	0.5178	0.0254

Table 15 Summary of large datasets

Data set	Cases	Features			Classes	Inputs	1-NN error
		C	B	N			
adult	48842	6	1	7	2	105	0.2033
krkopt	28056	6	–	–	18	6	0.4520
letter	20000	16	–	–	26	16	0.0390
magic	19020	10	–	–	2	10	0.1351
shuttle	58000	9	–	–	7	9	0.0015

In each replication the available data is partitioned into two random equal-sized sets. Each method is trained on one set at a time and tested on the other set.

For these problems we used the three standard methods, IB3, ICF and Drop3, CHC and our method. CHC was not able to scale well. For all the problems the results obtained were



(a) MSS (b) Entropy-based

(c) IMOEA (d) LVQPRU

Fig. 6 Summary of results in terms of testing error and storage for the methods

poor and the time needed far higher than the time spent by the other methods. Due to the size of the datasets we have evolved our model using 50 subpopulations, in order to keep a ratio of instances per subpopulation similar to the previous experiments, and the inner loop $N = 1$. The remaining parameters are the shown in Table 4. The results are shown in Table 16 for standard methods and in Table 17 for CCIS and CHC. The table shows a very good scalability of our method. Even for problems with more than 20000 instances, it is able to obtain better results on average than the standard methods. The achieved reduction is markedly better than the obtained by DROP3 and ICF, keeping at the same time a testing error better than ICF and very similar to DROP3. Only IB3 obtains similar results in terms of reduction, but with worse testing errors.

Regarding computational cost, Fig. 7 shows the average time spent by both evolutionary algorithms, CCIS and CHC, for the problems. The figure shows that CCIS is able not only to achieve better results but also in less time than CHC. This behavior corroborates the observed in the previous study of computational cost (see Fig. 5).

8 Discussion

In the previous sections we have presented a new approach to instance selection for instance based learning based on a cooperative coevolutionary model. This model has the interesting feature of being independent from the classifier used, in contrast with most current methods that are based on k-NN classifiers. Additionally, the flexibility of the evolutionary approach

Table 16 Summary of results the three standard methods in terms of testing error and storage requirements for large datasets

Dataset	DROP3		ICF		IB3	
	Storage	Error	Storage	Error	Storage	Error
adult	0.2908	0.2276	0.2454	0.2712	0.0816	0.2332
letter	0.6413	0.4496	0.6995	0.4321	0.4767	0.4410
krkopt	0.1476	0.0770	0.1497	0.1415	0.1402	0.0850
magic	0.1580	0.1104	0.2345	0.2576	0.0011	0.9832
shuttle	0.0037	0.0014	0.0333	0.0196	0.0033	0.0016
Average	0.2483	0.1732	0.2725	0.2244	0.1406	0.3488

Table 17 Summary of results for CCIS and CHC methods in terms of testing error and storage requirements for large datasets

Dataset	CCIS		CHC	
	Storage	Error	Storage	Error
adult	0.0565	0.1981	0.2081	0.2056
letter	0.2850	0.4663	0.5237	0.4711
krkopt	0.2371	0.0970	0.2125	0.1020
magic	0.0063	0.0889	0.2139	0.2129
shuttle	0.0302	0.0029	0.2929	0.0022
Average	0.1139	0.1731	0.2902	0.1988

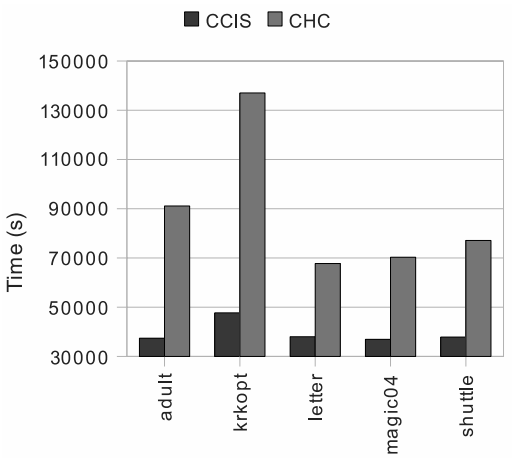


Fig. 7 Summary of average time spent by CCIS and CHC for large datasets

allows the user to put more emphasis on storage reduction or testing error modifying the relative weight of these two values in the fitness function. In this way, the algorithm can be better adapted to the necessities of the researcher.

As stated in the introduction we developed our method with two main objectives, establishing the validity of cooperative coevolution for instance selection and obtaining a method able to improve standard methods and with a good scalability. We believe that the results reported in this paper show that both objectives have been fulfilled.

We have carried out a thorough comparison of the proposed methodology with state-of-the-art methods in both storage requirements and testing error. We have used a large set of 50 benchmark problems with different features, sizes and numbers of classes and eight instance selection methods to compare with. The performed experiments show a clear advantage of the proposed method over standard algorithms. In terms of storage requirements, the improvement achieved by the cooperative method is specially marked.

An additional set of experiments has been carried out to test the behavior of the cooperative algorithm in the presence of noise in the labels of the instances. These experiments show that CCIS is able to maintain its advantage over the other algorithms in noisy data, even improving the differences with some of them.

An alternative version of the algorithm has been presented, which can be implemented distributedly to achieve a more efficient execution of the algorithm. Although this implementation is based on an approximate evaluation of the individuals, the experiments show that it can match the performance of the standard cooperative algorithm in terms of storage reduction. Nevertheless, the testing error of this implementation is significantly worse, as it should be expected, although the differences are not dramatic. Scalability to large datasets has been studied using five datasets. Our method has shown very good results even for those large datasets, achieving an overall performance better than the standard methods and CHC.

9 Future work

As future work, we are working on a natural extension of the model. As the evaluation of the selectors and combinations is made using different criteria, the application of a multiobjective genetic algorithm is natural instead of a weighted sum of the criteria. In a previous work (García-Pedrajas et al. 2002) we have shown how the multiobjective algorithm can outperform the non-multiobjective one in the field of artificial neural network evolution. Also, due to the excellent performance of IMOEA, subspace selection is a promising feature to be incorporated into our approach.

Furthermore, the results in this paper open a very interesting research line. The idea underlaying our model is that we can obtain good results for instance selection with a method that combines several partial views of the dataset, but that never has to deal with the whole dataset. This idea can be extended to scale up instance selection algorithms. In the same way we use it for constructing a more scalable evolutionary instance selection algorithm, we can use it to scale up other instance selection algorithms. On the other hand, trying to develop algorithms with a lower efficiency order is likely to be a fruitless search. Obtaining the nearest neighbor of a given instance is $O(n)$.[6] To test whether removing an instance affects the accuracy of the nearest neighbor rule, we must measure the effect on the other instances of the absence of the removed one. Measuring this effect involves recalculating, directly or indirectly, the nearest neighbors of the instances. The result is a process of $O(n^2)$. In this way, the attempt to develop algorithms of an efficiency order below $O(n^2)$ is not very promising. Thus, the method proposed in this paper offers a different and promising alternative.

In a certain sense, we can liken our approach to ensemble construction of classifiers. We are constructing a instance selection algorithm joining together several instance selection algorithms that have only a partial view of the dataset. These instance selection methods

[6] Although more sophisticated methods, like kd-trees, can be used, these methods are complex and increase memory requirements.

applied to subsets of instances are then similar to the weak classifiers in an ensemble. Thus, our approach shows a way to introduce concepts from classifier ensembles into instance selection. Furthermore, it can be extended to related problems, such as feature selection and clustering.

As a side result, the accomplished scalability of instance selection allows the application of this technique to new fields, such as Bioinformatics, where existing algorithms are too computationally costly to be used.

10 Conclusion

As an overall summary, we think that the major contribution of our work is showing that cooperative coevolution can be used to tackle large problems taking advantage of its inherently modular nature. In fact we are showing that a combination of cooperative coevolution together with the principle of divide-and-conquer can be very effective both in terms of improving performance and in reducing computational cost. Other problems with similar features, such as feature selection, can also benefit from this approach. Additionally, we have shown that the inherent modularity of cooperative coevolution makes it a good candidate for a distributed implementation allowing a more efficient execution of the algorithm.

As stated before, in the best case, existing instance selection algorithms are of a quadratic complexity (Cano et al. 2007) in the number of instances. For large problems these methods are not applicable. One natural way of scaling up a certain algorithm is dividing the original problem into several simpler subproblems and applying the algorithm separately to each subproblem. In this way, we might scale up instance selection dividing the original dataset into several disjoint subsets and performing the instance selection process separately on each subset. However, this method would suffer from the partial knowledge it has of the dataset. To evaluate the relevance of an instance, the algorithm needs to know the whole dataset, as that relevance depends on all the other instances. The results reported in this paper show a way to accomplish this divide and conquer approach using cooperative coevolution, and open an interesting field of research.

Acknowledgements The authors would like to acknowledge Dr. J.R. Cano for his valuable help. They would also like to acknowledge the anonymous reviewers for their valuable comments. This work was supported in part by Project TIN2008-03151 of the Spanish Ministry of Education and Project P07-TIC-02682 of the Junta de Andalucía.

References

Aha, D. W., Kibler, D., & Albert, M. K. (1991). Instance-based learning algorithms. *Machine Learning*, *6*, 37–66.

Anderson, T. W. (1984). *An introduction to multivariate statistical analysis* (2nd edn.). *Wiley series in probability and mathematical statistics*. New York: Wiley.

Baluja, S. (1994). *Population-based incremental learning* (Technical Report CMU-CS-94-163). Carnegie Mellon University, Pittsburgh.

Barandela, R., Ferri, F. J., & Sánchez, J. S. (2005). Decision boundary preserving prototype selection for nearest neighbor classification. *International Journal of Pattern Recognition and Artificial Intelligence*, *19*(6), 787–806.

Blum, A., & Langley, P. (1997). Selection of relevant features and examples in machine learning. *Artificial Intelligence*, *97*, 245–271.

Bongard, J., & Lipson, H. (2005a). Active coevolutionary learning of deterministic finite automata. *Journal of Machine Learning Research*, *6*, 1651–1678.

Bongard, J. C., & Lipson, H. (2005b). Nonlinear system identification using coevolution of models and tests. *IEEE Transactions on Evolutionary Computation*, *9*(4), 361–384.
Brighton, H., & Mellish, C. (2002). Advances in instance selection for instance-based learning algorithms. *Data Mining and Knowledge Discovery*, *6*, 153–172.
Brodley, C. E. (1995). Recursive automatic bias selection for classifier construction. *Machine Learning*, *20*(1/2), 63–94.
Cano, J. R., Herrera, F., & Lozano, M. (2003). Using evolutionary algorithms as instance selection for data reduction in KDD: an experimental study. *IEEE Transactions on Evolutionary Computation*, *7*(6), 561–575.
Cano, J. R., Herrera, F., & Lozano, M. (2005). Stratification for scaling up evolutionary prototype selection. *Pattern Recognition Letters*, *26*(7), 953–963.
Cano, J. R., Herrera, F., & Lozano, M. (2007). Evolutionary stratified training set selection for extracting classification rules with trade off precision-interpretability. *Data & Knowledge Engineering*, *60*(1), 90–108.
Chaudhuri, S., Motwani, R., & Narasayya, V. (1998). Random sampling for histogram construction: how much is enough? In L. Haas & A. Tiwary (Eds.), *Proceedings of ACM SIGMOD, international conference on management of data* (pp. 436–447). New York, USA.
Chellapilla, K., & Fogel, D. B. (1999). Evolving neural networks to play checkers without relying on expert knowledge. *IEEE Transactions on Neural Networks*, *10*(6), 1382–1391.
Chen, J. H., Chen, H. M., & Ho, S. Y. (2005). Design of nearest neighbor classifiers: multi-objective approach. *International Journal of Approximate Reasoning*, *40*(1–2), 3–22.
Cochran, W. (1977). *Sampling techniques*. New York: Wiley.
Cover, T. M., & Thomas, J. A. (1991). *Elements of information theory*. *Wiley series in telecommunication*. New York: Wiley.
Demšar, J. (2006). Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research*, *7*, 1–30.
Dietterich, T. G. (1998). Approximate statistical tests for comparing supervised classification learning algorithms. *Neural Computation*, *10*(7), 1895–1923.
Dietterich, T. G. (2000). An experimental comparison of three methods for constructing ensembles of decision trees: bagging, boosting, and randomization. *Machine Learning*, *40*, 139–157.
Eshelman, L. J. (1990). *The CHC adaptive search algorithm: how to have safe search when engaging in nontraditional genetic recombination*. San Mateo: Morgan Kauffman.
García-Pedrajas, N., & Fyfe, C. (2007) Immune network based ensembles. In *Neurocomputing* (pp. 1155–1166).
García-Pedrajas, N., & Ortiz-Boyer, D. (2007). A cooperative constructive method for neural networks for pattern recognition. *Pattern Recognition*, *40*(1), 80–99.
García-Pedrajas, N., Hervás-Martínez, C., & Muñoz-Pérez, J. (2002). Multiobjective cooperative coevolution of artificial neural networks (multi-objective cooperative networks). *Neural Networks*, *15*(10), 1255–1274.
García-Pedrajas, N., Hervás-Martínez, C., & Ortiz-Boyer, D. (2005). Cooperative coevolution of artificial neural network ensembles for pattern classification. *IEEE Transactions on Evolutionary Computation*, *9*(3), 271–302.
Gates, G. W. (1972). The reduced nearest neighbor rule. *IEEE Transactions on Information Theory*, *18*(3), 431–433.
Goldberg, D. E. (1989). *Genetic algorithms in search, optimization and machine learning*. Reading: Addison–Wesley.
Grefenstette, J. J. (1987). Incorporating problem specific knowledge into genetic algorithms. In L. Davis (Ed.), *Genetic algorithms and simulated annealing* (pp. 42–60). San Mateo: Morgan Kaufmann.
Hettich, S., Blake, C., & Merz, C. (1998). UCI Repository of machine learning databases. http://www.ics.uci.edu/~mlearn/MLRepository.html.
Hillis, W. D. (1991). Co-evolving parasites improves simulated evolution as an optimization procedure. In C. G. Langton, C. Taylor, J. D. Farmer, & S. Rasmussen (Eds.), In *Artificial Life II* (pp. 313–384).
Holland, J. H. (1975). *Adaptation in natural and artificial systems*. Ann Arbor: The University of Michigan Press.
Hussain, F., Liu, H., Tan, C., & Dash, M. (1999). *Discretization: an enabling technique* (Technical Report TRC6/99). School of Computing, National University of Singapore.
Ishibuchi, H., & Nakashima, T. (2000). Pattern and feature selection by genetic algorithms in nearest neighbor classification. *Journal of Advanced Computational Intelligence and Intelligent Informatics*, *4*(2), 138–145.
Kivinen, J., & Mannila, H. (1994). The power of sampling in knowledge discovery. In *Proceedings of the thirteenth ACM SIGACT-SIGMOD-SIGART symposium on principles of database systems* (pp. 77–85). Minneapolis, Minnesota, USA.

Kuncheva, L. (1995). Editing for the *k*-nearest neighbors rule by a genetic algorithm. *Pattern Recognition Letters*, *16*, 809–814.

Leung, Y. W., & Wang, Y. P. (2001). An orthogonal genetic algorithm with quantization for global numerical optimization. *IEEE Transactions on Evolutionary Computation*, *5*(1), 41–53.

Li, J., Manry, M. T., Yu, C., & Wilson, D. R. (2005). Prototype classifier design with pruning. *International Journal of Artificial Intelligence Tools*, *14*(1–2), 261–280.

Liu, H., & Motoda, H. (1998). *Feature selection for knowledge discovery and data mining*. Norwell: Kluwer.

Liu, H., & Motoda, H. (2002). On issues of instance selection. *Data Mining and Knowledge Discovery*, *6*, 115–130.

Louis, S. J., & Li, G. (1997). Combining robot control strategies using genetic algorithms with memory. In *Lecture notes in computer science: Vol. 1213*. *Evolutionary programming VI* (pp. 431–442). Berlin: Springer.

Michalewicz, Z. (1994). *Genetic algorithms + Data structures = Evolution programs*. New York: Springer.

Moriarty, D. E., & Miikkulainen, R. (1995). Discovering complex Othello strategies through evolutionary neural networks. *Connection Science*, *7*(3), 195–209.

Moriarty, D. E., & Miikkulainen, R. (1996). Efficient reinforcement learning through symbiotic evolution. *Machine Learning*, *22*, 11–32.

Potter, M., & De Jong, K. A. (1994). A coopetative coevolutionary approach to function optimization. In *Proceedings of the third conference on parallel problem solving from nature* (pp. 249–257).

Potter, M. A., & De Jong, K. A. (2000). Cooperative coevolution: an architecture for evolving coadapted subcomponents. *Evolutionary Computation*, *8*(1), 1–29.

Provost, F. J., & Kolluri, V. (1999). A survey of methods for scaling up inductive learning algorithms. *Data Mining and Knowledge Discovery*, *2*, 131–169.

Reeves, C. R., & Bush, D. R. (2001). Using genetic algorithms for training data selection in RBF networks. In H. Liu & H. Motoda (Eds.), *Instances selection and construction for data mining* (pp. 339–356). Norwell: Kluwer.

Ritter, G. L., Woodruff, H. B., Lowry, S. R., & Isenhour, T. L. (1975). An algorithm for selective nearest neighbor decision rule. *IEEE Transactions on Information Theory*, *21*(6), 665–669.

Rosin, C. D., & Belew, R. K. (1997). New methods for competitive coevolution. *Evolutionary Computation*, *5*(1), 1–29.

Smith, P. (1998). *Into statistics*. Singapore: Springer.

Smyth, B., & Keane, M. T. (1995). Remembering to forget. In C. S. Mellish (Ed.), *Proceedings of the fourteenth international conference on artificial intelligence* (Vol. 1. pp. 377–382).

Son, S. H., & Kim, J. Y. (2006). Data reduction for instance-based learning using entropy-based partitioning. In *Lecture notes in computer science: Vol. 3982*. *Proceedings of the international conference on computational science and its applications—ICCSA 2006* (pp. 590–599). Berlin: Springer.

Syswerda, G. (1991). A study of reproduction in generational and steady-state genetic algorithms. In G. Rawlins (Ed.), *Foundations of genetic algorithms* (pp. 94–101). San Mateo: Morgan Kaufmann.

Tomek, I. (1976). Two modifications of CNN. *IEEE Transactions on Systems, Man, and Cybernetics—Part B: Cybernetics*, *SMC-6*, 769–772.

Whitley, D. (1989). The GENITOR algorithm and selective pressure. In M. K. Publishers (Ed.), *Proc. 3rd international conf. on genetic algorithms* (pp. 116–121). Los Altos, CA.

Whitley, D., & Kauth, J. (1988). GENITOR: a different genetic algorithm. In *Proceedings of the Rocky mountain conference on artificial intelligence* (pp. 118–130). Denver, CO.

Whitley, D., & Starkweather, T. (1990). GENITOR II: a distributed genetic algorithm. *Journal of Experimental Theoretical Artificial Intelligence*, *2*, 189–214.

Wilson, D. L. (1972). Asymptotic properties of nearest neighbor rules using edited data. *IEEE Transactions on Systems, Man, and Cybernetics*, *2*(3), 408–421.

Wilson, D. R., & Martinez, A. R. (1997). Instance pruning techniques. In D. Fisher (Ed.), *Proceedings of the fourteenth international conference on machine learning* (pp. 404–411). San Francisco, CA, USA.

Wilson, D. R., & Martinez, T. R. (2000). Reduction techniques for instance-based learning algorithms. *Machine Learning*, *38*, 257–286.

Zitzler, E., Thiele, L., Laumanns, M., Fonseca, C. M., & Grunert, V. (2003). Performance assesment of multiobjective optimizers: an analysis and review. *IEEE Transactions on Evolutionary Computation*, *7*(2), 117–132.