Soft Comput (2012) 16:1267–1286
DOI 10.1007/s00500-012-0811-y

ORIGINAL PAPER

Clustering-based initialization of Learning Classifier Systems

Effects on model performance, readability and induction time

Fani A. Tzima · Pericles A. Mitkas · John B. Theocharis

Published online: 16 February 2012

Abstract The present paper investigates whether an "informed" initialization process can help supervised LCS algorithms evolve rulesets with better characteristics, including greater predictive accuracy, shorter training times, and/or more compact knowledge representations. Inspired by previous research suggesting that the initialization phase of evolutionary algorithms may have a considerable impact on their convergence speed and the quality of the achieved solutions, we present an initialization method for the class of supervised Learning Classifier Systems (LCS) that extracts information about the structure of studied problems through a pre-training clustering phase and exploits this information by transforming it into rules suitable for the initialization of the learning process. The effectiveness of our approach is evaluated through an extensive experimental phase, involving a variety of real-world classification tasks. Obtained results suggest that clustering-based initialization can indeed improve the predictive accuracy, as well as the interpretability of the induced knowledge representations, and paves the way for further investigations of the potential of better-than-random initialization methods for LCS algorithms.

Keywords Learning Classifier Systems (LCS) · Supervised Learning · Classification · Initialization

F. A. Tzima (✉) · P. A. Mitkas · J. B. Theocharis
Department of Electrical and Computer Engineering,
Aristotle University of Thessaloniki,
541 24 Thessaloniki, Greece
e-mail: fani@issel.ee.auth.gr

P. A. Mitkas
e-mail: mitkas@eng.auth.gr

J. B. Theocharis
e-mail: theochar@eng.auth.gr

1 Introduction

In real-world classification problems, it is often the case that the desired solution must be interpretable by human experts and/or decision makers. This is especially true when the extracted knowledge is to be used in the medical domain (Holmes and Sager 2005), for public information provision and, more importantly, for decision-making support, such as in the air quality forecasting domain (Tzima et al. 2011). For tackling these kinds of problems, where a "crystal clear solution" (Lanzi 2008), rather than just a solution, is required, there are two options:

1. we can either apply an already available "black-box" method, and later find a way to gain insight into the solution and/or let interested parties "look inside the box" (Breiman 2002), or
2. we can opt from the beginning for methods inducing intuitive representations that inherit the basic characteristics of the knowledge domain they are mapping.

Methods that belong to the second approach and may provide an effective and computationally feasible alternative, in line with our initial requirement for high interpretability, include (a) *classifiers inducing sets of rules*, such as FOIL (Quinlan 1996), PART (Frank and Witten 1998), HIDER (Aguilar-Ruiz et al. 2003, 2007), and SIA (Venturini 1993); (b) *learning classifier systems*, such as XCS (Wilson 1995), UCS (Bernadó-Mansilla and Garrell-Guiu 2003; Orriols-Puig and Bernadó-Mansilla 2008b), GAssist (Bacardit 2004) and ILGA (Guan and Zhu 2005); and (c) *algorithms inducing decision trees*, such as C4.5 (Quinlan 1993).

The scope of our current work comprises offline classification problems where, hopefully without loss of generality in terms of applicability to other LCS, we choose to

study supervised LCS learners following the "Michigan approach". Such LCS maintain a cooperative population of condition-action rules, termed classifiers, and combine supervised learning (Mitchell 1997) with a genetic algorithm (GA) to solve problems. The GA works on classifier conditions in an effort to adequately decompose the target problem into a set of subproblems, while supervised learning evaluates classifiers in each of them (Lanzi 2008). Although the most prominent example of this class of systems is UCS (Bernadó-Mansilla and Garrell-Guiu 2003; Orriols-Puig and Bernadó-Mansilla 2008b), an accuracy-based LCS, we have recently introduced SS-LCS, a supervised strength-based LCS that departs from the accuracy-based approach to fitness computation (Tzima and Mitkas 2010). SS-LCS provides an efficient and robust alternative for offline classification tasks by extending previous strength-based frameworks (Wilson 1994; Bonelli et al. 1990; Kovacs 2002a; b) and, together with UCS, will serve as the basis for our current investigation.

Although recent years have seen a shift of interest toward applying LCS to single-step decision tasks, such as pattern recognition and predictive data mining (DM), the high competence of LCS has been impaired to some extent by (i) the large number of semantic-free rules that are evolved (Orriols-Puig et al. 2009), and (ii) the long processing times required for their evolution. These shortcomings can be attributed not only to the rule representations traditionally used by LCS learners and their limited expressiveness for certain tasks, but also to the, arguably slow, evolutionary process involved in their search component. Several approaches seek to alleviate the problem of large rulesets by employing rule reduction techniques after and/or during learning (Wilson 2002; Dixon et al. 2003) or introducing linguistic fuzzy representations of rules—see (Orriols-Puig et al. 2009) for an example of fuzzy Michigan LCS, (Ishibuchi et al. 2005) for a fuzzy Pittsburgh LCS, and (González and Pére 1999) for a fuzzy Iterative Rule Learning (IRL) algorithm. However, the problem of large rulesets has also been linked to the failure of the GA process to effectively generalize over the search space of correct rules. This observation, together with the rather long execution times required by LCS learners, points to problems that have already been identified and addressed in the GA literature, outside the LCS domain.

More specifically, although theoretical developments by Holland (1975) and De Jong (1975) have long laid the foundations of GAs (and therefore LCS), and the use of "nonstandard" evolutionary algorithms—involving tailored representations, operators, etc.—has greatly increased the range of problems/domains to which evolutionary algorithms can be effectively applied, there still remain several problems that stem from the very nature of evolutionary search methods. These problems include *premature convergence* to suboptimal or undesired solutions, low *convergence speed* that results in poor evolution, and large *execution times* due to mass computations of evaluating fitnesses and/or applying genetic operators.

Most of the work in the direction of alleviating these problems has focused on introducing new or improving existing selection mechanisms and genetic operators, adaptive control of parameter settings, etc. In the meantime, in cases where no external information is available about the solution, random initialization has been the most commonly used method to generate initial populations. Although it is recognized that population initialization can affect not only convergence speed, but also the quality of the final solution, surprisingly little research is reported in this field, with existing work being primarily in the domains of function optimization, scheduling, and case-injected GAs (see Sect. 2 for more details). The idea of using heuristics to choose better-than-random individuals for the initial population (that may lead to significantly faster convergence to a good solution or, alternatively, a better solution in the same amount of time) has not, to our knowledge, been applied in Michigan LCS research. Most systems following the Michigan approach avoid the *pre-training* initialization stage and start with empty rulesets that are progressively populated through the use of a covering operator *during learning*.

Aiming to explore the idea of better-than-random pre-training initialization for the class of supervised LCS algorithms, we propose an initialization procedure that extracts "summary information" from the original problem description (i.e., the training dataset) and processes it to produce a set of rules for the initialization of the training phase. This process starts with clustering the training dataset on a per-class basis and uses the discovered clusters (their centroids, as well as the corresponding cluster assignments of the training instances) to create a small number of diverse rules with various degrees of generalization that get injected into the initial population of the GA. The rest of the ruleset is still populated using the initialization mechanism employed by all Michigan-style LCS algorithms, namely the covering operator. Thus, the GA begins its search from this combined, and hopefully more competent, initial population.

Our experimental procedure designed to validate the effectiveness of the proposed method involves a comparative analysis of the clustering-based initialization (CI) component's performance against the baseline approaches of SS-LCS and UCS with no pre-training initialization. The corresponding results—obtained from two sets of experiments, studying the predictive accuracy and the readability of evolved models, respectively—reveal the potential of the *pre-training clustering-based initialization process* for

supervised LCS and are readily extensible to other Michigan-style LCS.

The remainder of this paper is structured as follows: Sect. 2 outlines related work, while Sect. 3 provides a high-level description of the two studied LCS algorithms, namely SS-LCS and UCS, for supervised classification tasks. Section 4 presents our proposed clustering-based initialization component and analyzes how it results in rules suitable for the initialization of the studied LCSs' exploration process. Our experimental methodology follows in Sect. 5, along with the results obtained from experiments, organized in two distinct sets, applying the two versions (with and without the CI component) of SS-LCS and UCS, as well as their rival algorithms, to 20 real-world classification tasks. Section 6 presents an initial investigation of the variability of results achieved with the CI method, due to specific design choices. Finally, Sect. 7 concludes this paper by summarizing our presented work, restating our contributions and conclusions, and identifying future research directions.

2 Related work

The use of operators that leverage training instances during the process of creating rules is not new in the LCS literature. The covering operator, already introduced in ZCS (Wilson 1994), and later traditionally used by most Michigan-style LCS reported in the literature—see XCS (Wilson 1995) and its descendants, UCS (Bernadó-Mansilla and Garrell-Guiu 2003; Orriols-Puig and Bernadó-Mansilla 2008b), etc.—is based on exactly this principle: the system creates rules as generalized versions of its training instances, whenever it cannot match a provided example (or, in certain systems, when it has zero probability of correctly classifying it, due to an empty correct set). It is important to note, though, that the covering operator progressively fills the ruleset, being invoked only when the system does not already have a rule matching (or in some cases correctly classifying) a specific instance, and applies *during* the learning phase, that is in close interaction with the evolutionary search and the deletion processes employed in LCS. To our knowledge, there has not been any work in the literature that investigates *pre-training* initialization procedures for Michigan-style LCS. However, a significant amount of research efforts has been invested in the facetwise analysis of such systems, including (but not limited to) the behavior and effects of the covering process and the tuning of the system's generalization probability (or equivalently, specificity) to ensure proper population initialization and adequate fitness evaluation of individuals (Butz et al. 2004).

In the wider GBML research area, an initialization operator, similar to the covering process of LCS, is used in the HIDER system (Aguilar-Ruiz et al. 2003) that iteratively evolves hierarchical classification rules. In this case, however, the operator, that creates rules as generalized versions of randomly sampled training instances, is applied prior to training to initialize the GA's population. Further examples of initialization operators based on training instances may be found in systems following the Pittsburgh approach, such as the GIL system (Janikow 1992) that employed "mixed initialization", with part of the population being initialized randomly, while the rest of the rules are created as exact copies of (randomly) sampled training instances. A more elaborate approach is chosen by the Pittsburgh-style LCS GAssist (Bacardit 2005): several policies are tested for (i) creating rules by generalizing a sample of the training instances and (ii) tuning the instance-sampling probability of the system, with the ultimate goal of identifying a robust initialization setting to be used as the default initialization policy of the system.

In areas other than Machine Learning, GA initialization has been sparsely studied in the domains of task scheduling, function optimization, and case-injected GAs. In the scheduling domain, where speed is often a key issue, several authors have investigated heuristic initialization. Several stochastic heuristics, aimed at generating competent initial solutions for the problem of timetabling, are outlined by Burke et al. (1998). The authors verify that heuristics with optimal degrees of randomness can produce high-fitness individuals without sacrificing diversity. In another paper by Zhang et al. (2011), where a GA is proposed for solving the flexible job-shop scheduling problem (FJSP), the Global Selection (GS) and Local Selection (LS) methods are designed to generate high-quality initial populations and, thus, improve the convergence speed and the quality of final solutions.

Other efforts in the field of (unimodal and multimodal) function optimization include opposition-based population initialization in Differential Evolution (Rahnamayan et al. 2007), quasi-random population initialization (Maaranen et al. 2004), and the uniform and unbiased initialization methods (Chou and Chen 2000). In all cases, the authors report improvements in the performance of GAs, in terms of the quality of final solutions and/or the algorithms' convergence speed. Additional studies on initialization methods can be found in the domains of GA-based combinatorial optimization (Kang and Jung 2006), optimization of binary problems (Kallel and Schoenauer 1997), anytime learning (Ramsey and Grefenstette 1993), and case-injected learning (Louis and McDonnell 2004). The latter approach, named the Case Injected Genetic AlgoRithm (CIGAR), inserts a small number of solutions from previously encountered problems into the initial population of

the GA, while maintaining diversity by randomly initializing the rest of it.

3 Supervised Learning Classifier Systems

As already mentioned, the potential and applicability of our proposed CI method is validated, through its incorporation into two algorithms from the class of supervised Michigan-style LCS, namely SS-LCS and UCS. Thus, before presenting the CI method in detail, the following Sections provide a high-level description of the studied LCS algorithms, especially focusing on the not so well-known SS-LCS algorithm.

3.1 The Supervised Classifier System (UCS)

UCS is an accuracy-based learning classifier system introduced in Bernadó-Mansilla and Garrell-Guiu (2003) that inherits the primary features of XCS, but specializes them for supervised learning tasks. More specifically, the performance component is adjusted to a supervised learning scheme that focuses the exploration on consistently correct rules, rather than consistently accurate ones as in XCS. UCS also computes the accuracy (acc) of classifiers as the percentage of their correct classifications and employs fitness sharing (Orriols-Puig and Bernadó-Mansilla 2008b), such that the fitness F_{cl} of a classifier cl is computed by the following equation:

$$F_{\mathrm{cl}} \leftarrow F_{\mathrm{cl}} + \beta(k'_{\mathrm{cl}} - F_{\mathrm{cl}}) \tag{1}$$

where β is an algorithm parameter and k'_{cl} is computed according to

$$k'_{\mathrm{cl}} = \frac{k_{\mathrm{cl}} \cdot \mathrm{num}_{\mathrm{cl}}}{\sum_{\mathrm{cl}_i \in [M]} k_{\mathrm{cl}_i} \cdot \mathrm{num}_{\mathrm{cl}_i}}$$

with $\mathrm{num}_{\mathrm{cl}}$ being the classifier's numerosity, that is its number of copies in the ruleset, and k_{cl} its relative accuracy ($k_{\mathrm{cl}\in[!C]} = 0$):

$$k_{\mathrm{cl}\in[C]} = \begin{cases} 1 & \text{if } \mathrm{acc} > \mathrm{acc}_0 \\ a(\mathrm{acc}/\mathrm{acc}_0)^v & \text{otherwise} \end{cases}$$

3.2 The Strength-based Supervised Classifier System (SS-LCS)

SS-LCS is a strength-based supervised learning classifier system, introduced in Tzima and Mitkas (2010), that departs from the reinforcement learning approach to classifier evaluation traditionally used in LCS (Bonelli et al. 1990; Wilson 1994, 1995; Butz and Wilson 2001; Kovacs 2002a, b) and bases its fitness on more straightforward DM-based rule performance metrics. SS-LCS calculates fitness by directly estimating a classifier's payoff rate per step (that can be either positive or negative) from its strength value and is a best action map (BAM) learner that focuses on the evolution of consistently correct rules, leaving the consistently incorrect, though accurate, ones out of the exploration process.

The following is a more detailed description of the SS-LCS algorithm's components that are necessary for our current investigation of single-step classification tasks.

3.2.1 Classifier parameters

SS-LCS employs a population **P** of gradually evolving, cooperative classifiers (rules) that collectively form the solution to the target classification task, with each encoding a fraction of the problem domain. Associated with each classifier, is a number of parameters:

1. the numerosity *num* is the number of the classifier's copies (or microclassifiers) currently present in the ruleset;
2. the niche set size *ns* estimates the average size of the correct sets the classifier has participated in;
3. the time step *ts* of the last occurrence of a GA in a correct set the classifier has belonged to;
4. the experience *msa* that is measured as the classifier's number of appearances in match sets;
5. the number of the classifier's correct and incorrect decisions, *tp* and *fp* respectively;
6. a scalar strength value *str* that estimates the classifier's average received reward per step; and
7. the fitness *F* that is a measure of the classifier's quality.

3.2.2 Performance component

At each discrete time-step t during learning (or testing), SS-LCS receives a binary encoded instance vector V_t along with its associated class $c_t(V_t \rightarrow c_t)$, scans the current population of classifiers for those whose condition matches the input, and forms the *matching set* **M**. Next, the *correct set* **C** is formed, containing all members of **M** advocating the correct action c_t, while the rest of the classifiers in **M**—the ones predicting classes other than c_t—are placed in the *incorrect set* **!C**. Finally, an action (classification decision) α is selected among those advocated by rules in **M**.

Depending on the inference strategy chosen for the particular problem, action selection may be *deterministic*, with the action advocated by the fittest classifier being selected, or based on a (possibly fitness-proportional) *voting* process among the classifiers advocating it in **M**. The vote of young classifiers (i.e., classifiers with $\mathrm{exp} < \theta_{\mathrm{del}}$) may also be decreased proportionally to their experience to prevent the votes of poorly evaluated classifiers in the

ruleset from disrupting the decision process when more experienced ones exist.

The *covering operator* is activated only in training (or explore) mode, when (i) the match set **M** is empty, (ii) the decision produced by the system (based on a non-empty match set **M**) is incorrect, or (iii) the correct set **C** is empty. Covering creates a new classifier with an action part equal to the current input's class c_t and a condition part matching V_t and generalized with a given probability $P_\#$ per locus.

It is also important to note that, under test mode, the population of SS-LCS does not undergo any changes; that is, the update, covering, and search mechanisms are disabled.

3.2.3 Update component

In training mode, each classification of a data instance is associated with an update of the matching classifiers' parameters. All classifiers in **M** increase their experience *msa* by one and all classifiers in **C** have their *ns* value updated to the arithmetic average of the sizes of all correct sets they have participated in so far. All classifiers in **M** also have their strength *str* and fitness *F* values updated, such that classifiers in **C** get their strength and fitness values increased, while the ones in **!C** decreased.

More specifically, the strength $\mathrm{str}_{\mathrm{cl}}$ of classifier cl is updated upon successful classification, according to:

$$\mathrm{str}_{\mathrm{cl}}^{(t)} = \mathrm{str}_{\mathrm{cl}}^{(t-1)} + \frac{R}{|C|^{(t)}} \tag{2}$$

where R is the reward apportioned to the system for correctly classifying an instance and $|C|^{(t)}$ is the size (in microclassifier terms) of the correct set **C** the classifier has participated in at step t. The strength of a classifier is also updated in case of a misclassification according to

$$\mathrm{str}_{\mathrm{cl}}^{(t)} = \mathrm{str}_{\mathrm{cl}}^{(t-1)} - p \cdot \frac{R}{ns_{\mathrm{cl}}^{(t)}} \tag{3}$$

where $ns_{\mathrm{cl}}^{(t)}$ is the classifier's average correct set size at step t. In other words, according to Eq. 3 a classifier's strength upon a misclassification is decreased by p times ($p \geq 1$) the average (positive) reward it has received so far.

Finally, the fitness $F_{\mathrm{cl}}^{(t)}$ of a classifier cl at any given timestep t is calculated according to

$$F_{\mathrm{cl}}^{(t)} = \mathrm{num}_{\mathrm{cl}}{}^{(t)} \cdot \frac{\mathrm{str}_{\mathrm{cl}}^{(t)}}{\mathrm{msa}_{\mathrm{cl}}{}^{(t)}} \tag{4}$$

Notice that instead of learning the fitness value via a temporal difference approach, SS-LCS directly estimates it by calculating the (possibly negative) *reward rate per step*, i.e., by dividing a classifier's strength $\mathrm{str}_{\mathrm{cl}}$ by its experience $\mathrm{msa}_{\mathrm{cl}}$. Equation 4 also factors in the classifier's numerosity $\mathrm{num}_{\mathrm{cl}}$, which is essential for the sharing scheme to effectively distribute reward at the microclassifier level.

3.2.4 Discovery component

SS-LCS employs a *steady-state niche genetic algorithm* applied on correct sets **C** that is invoked at a rate θ_{GA}, approximating the intervals needed for classifier fitnesses to settle to steady-state values. Thus, θ_{GA} is defined as a (minimum) threshold on the average time since the last GA invocation of the classifiers in **C**.

The evolutionary process employs parent selection based on tournaments of size $\tau_s = r \cdot |\mathbf{C}|$, with $r \in (0, 1)$ (Butz et al. 2005). Two parent classifiers are selected based on their fitness and copied to form two offspring after crossover, and mutation operators have been applied to them with given probabilities (χ and μ, respectively).

Before insertion into the classifier population, the offspring are checked for *subsumption* against each of their parents. If either of the parents is sufficiently experienced, *accurate* and more general than the offspring, the latter is not introduced into the population, but the parent's numerosity *num* is increased by one instead. If the offspring are not subsumed by either of their parents, they are introduced into the population and deletion is applied, if necessary, in order to maintain a constant population size |**P**| at the microclassifier level.

As subsumption is applied only when parents are accurate enough, it is essential that we provide a definition of *"accuracy"* in the SS-LCS framework. This definition, in line with our supervised approach, is expressed in DM terms and is independent of the fitness calculation scheme employed: a classifier cl in SS-LCS is considered *accurate* if its true positive rate ($\mathrm{tp}_{\mathrm{cl}}/\mathrm{msa}_{\mathrm{cl}}$) is greater than a threshold value $\mathrm{tpr}_{\mathrm{min}} \in (0, 1]$, with the actual value of $\mathrm{tpr}_{\mathrm{min}}$ being usually set close to 1.

3.2.5 Deletion

Unlike parent selection, deletion is applied on the whole population and is based on tournaments of size τ_d. The deletion probability of a classifier is proportional to the average size of the correct sets *ns* it has participated in and inversely proportional to its fitness, provided that the classifier is sufficiently experienced ($\mathrm{msa} > \theta_{\mathrm{del}}$), thus protecting newly created classifiers.[1] Given that the system maintains a record of the number of classifier matches per iteration through the dataset, higher deletion probabilities

[1] The deletion scheme employed is adapted from the one reported in Kovacs (1999).

are also assigned to classifiers not matching any instances in the training dataset.

4 The clustering-based initialization component

Inspired by relevant findings in the area of GA-based rule-induction and function optimization (Sect. 2), we developed a novel initialization method, named the *Clustering-based Initialization* (CI) method, that is applicable to any supervised Michigan LCS framework, provided that appropriate extensions are made to match possible rule representation differences. The CI method is applied *prior to training* and complements the covering operator, traditionally used by Michigan-style LCS *during training*, in providing competent initial solutions to the GA-based search component.

Clustering-based initialization is based on the idea that starting from a non-random set of rules may help the evolutionary process focus on the search-space optima (the optimal set of rules for the given classification task in our case) more effectively and quickly. Intuitively, this non-random set of initial rules should be based on any available information on the target problem, i.e., the training dataset,[2] and provide an effective "summary" of the knowledge available in it.

The CI method tries to leverage the potential of clustering algorithms to provide a representative set of points (centroids) for a given dataset. Given this set of centroids, we proceed by transforming them into rules suitable for the initialization of LCS, with the ultimate goal of boosting their performance not only in terms of predictive accuracy, but also in terms of training times—through the reduction of the evolutionary process' execution time—and readability.

The design of an effective clustering-based initialization process calls for answers—at both design and execution times—to a number of important questions:

Q1. What should be the clustering algorithms used?
Q2. Given the fact that most clustering algorithms require the a priori determination of the number of clusters to be created (or an equivalent parameter), how should this value be chosen? Should the number of clusters depend on the number of instances available in the training dataset and/or their distributions per class?
Q3. Upon completion of the clustering procedure, should all clusters be used for creating rules? Should we exclude overly sparse clusters?
Q4. Which method should be used to transform clusters and their centroids' attribute values into conditions for the corresponding rules? Should conditions be "centered" around centroid values? What is the appropriate amount of randomness for the conditions' generalization phase?
Q5. How should the action (class decision) of clustering-based rules be determined?
Q6. Given the fact that rule fitness (either strength- or accuracy-based) is central to all LCS algorithms' workings, what should be the initial fitness of clustering-based rules? Should we use a single initial fitness value or should we devise a method to "estimate" the potential of clustering-based rules and formulate their fitness values accordingly?
Q7. How should we evaluate the method's overall effectiveness? Given that we are interested in improving *convergence speed*, the *quality* (in terms of the overall predictive accuracy of the induced ruleset) and the *interpretability* (in terms of the ruleset size) of the final solution, what should be the experimental methodology used to compare random and clustering-based initialization?

In our current investigation, we chose to provide answers to the aforementioned questions based on the simplest possible strategy. Thus, we use the well-known k-means algorithm and, more specifically, its implementation provided by the machine learning tool WEKA (Witten and Frank 2005) [Q1]. We cluster the training dataset on a per-class basis, that is we cluster instances of each class separately, and set the number of clusters to be created to 20% of each class's prevalence in the training dataset [Q2]. After the clustering procedure, we use all clusters for creating rules [Q3]. For each cluster, a process utilizing information from the corresponding centroid and the instances assigned to the cluster is employed to create a rule. This process, termed the "Create Condition Part" process, works on creating the condition part of rules, whose class labels are set according to the partition of the data used in the clustering process that resulted in their "prototype" centroid [Q5].

The "Create Condition Part" process (explained in detail in Sect. 4.1) discriminates between numeric and nominal attributes when producing the corresponding conditions. However, it uniformly applies a generalization component to all final conditions, irrespective of whether they refer to nominal or numeric attributes, that is equivalent to the one employed in the "covering" process of LCS [Q4].

Finally, the initial parameters (strength/accuracy and fitness) of a centroid-based rule are calculated by the formulas

[2] We assume that no expert knowledge on the classification task and/or its solution is available at the time of learning.

$$p_{ij} = p_{\text{init}} \cdot (1 + \text{counts}_{ij}/|I_j|)$$

and

$$F_{ij} = f(p_{ij})$$

where p_{ij} is either the strength or the accuracy of the rule (depending on whether the employed LCS is strength- or accuracy-based), F_{ij} is the rule's fitness, p_{init} is a user-defined parameter of the LCS algorithm, f is the function used to compute fitness from strength (Eq. 4) or accuracy (Eq. 1), counts_{ij} is the number of instances of class j belonging to cluster i, and $|I_j|$ is the total number of instances of class j present in the training dataset.

The experimental methodology [Q7], employed to evaluate the method's overall effectiveness, is presented in Sect. 5 and involves conducting two series of experiments to systematically compare random and clustering-based initialization. As already mentioned, this comparison is based on the incorporation of the CI component into two supervised LCS algorithms, namely SS-LCS and UCS.

4.1 Leveraging clustering information of the target problem to produce initial rulesets

Before detailing the process of converting clusters into rules suitable for the initialization of our studied LCS, we provide a short overview of the rule representation employed therein.

Rules in both SS-LCS and (our implementation of) UCS follow the traditional production system form of "IF *conditions* THEN *action*". Given the fact that both algorithms aim at classification tasks, the action part is simply a class label, from the class attribute's set of possible values. On the other hand, the rule condition part consists of a conjunction of predicates that may take various forms, depending on the type of attributes (nominal or numeric) present in the training dataset and the representation chosen for the current run. Associated with each condition, there is an activation bit responsible for switching the condition *on* or *off* through genetic evolution or the built-in generalization process.

More specifically, for a *nominal attribute* x with possible values in the set $\mathbf{X} = \{V_1, V_2, \ldots, V_N\}$, the corresponding condition takes the form $x \in \mathbf{X_{sub}}$, where $\mathbf{X_{sub}} \subseteq \mathbf{X}$. For *numeric attributes* we employ the *interval representation*, according to which conditions are of the form $y \in [V_{\text{low}}, V_{\text{high}}]$, where V_{low} and V_{high} are real-valued.

Returning to the pre-training initial clustering of the target dataset, we report the steps included in the process in the following list:

1. The training dataset is partitioned into N subsets, where N is the number of classes, with $Partition_i$ including all instances of class_i present in the dataset.
2. For each $Partition_i$, $1 \leq i \leq N$
   (a) its instances are clustered into $M_i = \lceil \gamma \cdot |Partition_i| \rceil$ clusters, where $|Partition_i|$ is the number of instances in the ith partition, and γ is a user-defined parameter (set to 0.2 for our current study).
   (b) For each $cluster_j$, $1 \leq j \leq M_i$, identified in Step (2a), a rule is created, whose condition part is the result of applying the "Create Condition Part" process on the corresponding cluster:

$$RuleConditionPart_{ij} = CreateConditionPart(\ldots)$$

and its decision is set to the current class label i.

3. All $K = \sum_{i=1}^{N} M_i$ rules of the form

$$Rule_{ij} = RuleConditionPart_{ij} \rightarrow Class_i$$

created by clustering the training dataset are merged to create the ruleset used to initialize the learning process.

The "Create Condition Part" process, employed in Step (2b) above, is presented in Algorithm 1. Its inputs are the result of processing each discovered cluster and identifying

- the centroid, along with its values for each available attribute (centroid.values is a vector of size N_{attr});
- the minimum and maximum values for each numeric attribute based on the instances assigned to the cluster (cluster.minValues and cluster.maxValues are vectors of size N_{attr} that have "non-empty" values only in the indices corresponding to numeric attributes); and
- the labels of each nominal attribute present in instances assigned to the cluster (cluster.nomValues is a vector of size N_{attr}, whose items correspond to lists of varying size—zero for numeric attributes).

The process uses the aforementioned information to create a single condition per attribute and finally produces the "rule condition part" as their conjunction. Irrespective of the type of the attribute (numeric or nominal) involved, a condition may be completely removed (line 1) with probability $P^{ci}_{\#A}$. For *numeric attributes*, the condition coincides with the interval defined by the minimum and maximum values of the attribute, for instances assigned to the cluster (lines 1–1). For *nominal attributes*, the condition always includes the "mean" cluster value (identified as its centroid's value for the attribute—line 1) and may also include values present in the instances assigned to the cluster (line 1) with probability $(1 - P^{ci}_{\#Vp})$ and/or other values from the attribute's label domain with probability $(1 - P^{ci}_{\#Vnp})$ (line 1).

Algorithm 1 The 'Create Condition Part" process.
Used for transforming centroids, identified in the pre-training clustering phase, into rules for the initialization of LCS.

```
Input: centroid.values, cluster.nomValues,
       cluster.minValues, cluster.maxValues
Output: RuleConditionPart
        (conjunction of conditions, one per attribute)

START CreateConditionPart(...)
for k = 1 to N_attr do
  if Math.random() < P^ci_#A then
    Switch activation bit of condition k off
  else
    Switch activation bit of condition k on
  end if
  ---------- NOMINAL ATTRIBUTES
  if attribute_k is nominal then
    ValuesSet := ∅
    ValuesSet := ValuesSet ∪ centroid.values[k]
    for all values of attribute_k do
      if currentValue ∈ cluster.nomValues[k] then
        if (Math.random() ≥ P^ci_#Vp) then
          ValuesSet := ValuesSet ∪ currentValue
        end if
      else
        if (Math.random() ≥ P^ci_#Vnp) then
          ValuesSet := ValuesSet ∪ currentValue
        end if
      end if
    end for
    Create condition k as: attribute_k ∈ ValuesSet
    ---------- NUMERIC ATTRIBUTES
  else
    low_val = cluster.minValues[k]
    high_val = cluster.maxValues[k]
    Create condition k as:
      attribute_k ∈ [low_val, high_val]
  end if
  Add condition k to the RuleConditionPart
end for
END CreateConditionPart(...)
```

More formally, the possible condition parts for a nominal attribute x that can take one of a finite number of possible values in the set $\mathbf{X} = \{V_1, V_2, \ldots, V_N\}$ are of the form $x \in \mathbf{S}$, where $\mathbf{S} \subseteq \mathbf{X}$, with probability $(1 - \mathrm{P}^{ci}_{\#A})$, or otherwise (probability $\mathrm{P}^{ci}_{\#A}$) non-existent. Additionally, given two Boolean functions *IsCentroidValue*(V_i, C) and *IsPresentValue*(V_i, C) that check if value V_i is a match to the clusters C centroid value and if any instance assigned to the cluster C has the value V_i for attribute x, respectively, we may infer the following statements that hold for set **S**:

- $\mathrm{P}(V_i \in \mathbf{S} \mid \mathit{IsCentroidValue}(V_i, C)) = 1$
- $\mathrm{P}\big(V_j \in \mathbf{S} \mid \mathit{IsPresentValue}(V_j, \mathrm{C})\ \&$ $!\mathit{IsCentroidValue}(V_j, \mathrm{C})) = \mathrm{P}^{ci}_{\#Vnp}$
- $\mathrm{P}(V_k \in \mathbf{S} \mid \mathit{IsPresentValue}(V_k, \mathrm{C})\ \&$ $!\mathit{IsCentroidValue}(V_k, \mathrm{C})) = (1 - P^{ci}_{\#Vp})$
- $\mathrm{P}(\mathbf{S} \equiv \mathbf{X}) = (1 - P^{ci}_{\#Vnp}) \cdot P^{ci}_{\#Vp}$

Notice that the last statement refers to a condition entailing all of the attribute's possible values ($\mathbf{S} \equiv \mathbf{X}$), which is essentially equivalent to [*no condition*].

A simple example of the "Create Condition Part" process, entailing two attributes, one nominal and one numeric, is depicted in Fig. 1. Given the centroid of the discovered cluster, the corresponding limit values for the numeric attribute (values low_1 and $high_1$ on the X-axis), and the existing labels of the nominal attribute based on cluster assignments (values val_{2A} and val_{2C} corresponding to the "grayed" areas of the graph), the two lower boxes list the possible conditions for each attribute along with their probabilities of being created.

Based on the two lists, there are eight possible condition parts for the rule to be created. One of the most specific possible forms, created with $(1 - \mathrm{P}^{ci}_{\#A})^2 \cdot (1 - \mathrm{P}^{ci}_{\#Vnp}) \cdot \mathrm{P}^{ci}_{\#Vp}$ probability, would be:

$$\mathit{attribute}_1 \in [low_1, high_1]\mathrm{AND}$$
$$\mathit{attribute}_2 \in \{val_{2C}, val_{2B}\} \rightarrow \mathit{Cluster}_i\mathit{Class}$$

Notice that the decision part of the created rule is set to the class of the instances (subset of the initial training set), whose processing yielded the cluster in question.

5 Experimental validation of the clustering-based initialization method for LCS

5.1 Experimental methodology

The experimental part of our investigation has a threefold goal, as we aim at designing and conducting a battery of tests that would allow us to validate whether our proposed initialization approach can (i) boost the predictive accuracy of LCS (in our case SS-LCS and UCS), (ii) lead to the induction of more compact, and thus more understandable, knowledge models, and (iii) achieve any of the two aforementioned improvements in reduced training times (i.e., in less learning iterations than the baseline approach). Moreover, in all cases, we are interested in how the achieved results compare against those of other well-known approaches to classification rules' induction and, therefore, extend the experimental comparison to include models built with the GBML algorithms GAssist, HIDER and SIA, and the non evolutionary ML algorithms C4.5 and PART.

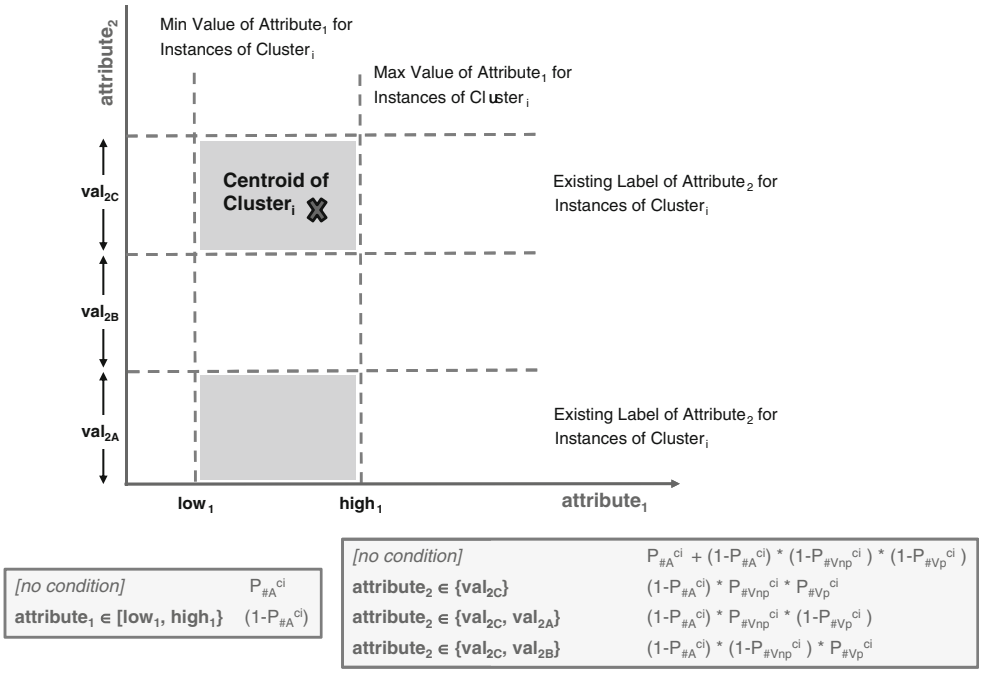


Fig. 1 Transformation of a cluster discovered in the clustering-based initialization phase into rule conditions. The overall condition part consists of two predicates, one from each of the lists in the *lower boxes*: the *lower left box* refers to the numeric attribute$_1$, while the *lower right* one to the nominal attribute$_2$

Given these requirements, and in the direction of validating our aforementioned hypotheses, this section reports and discusses the results of two sets of experiments. The *first one*, primarily aimed at studying performance through predictive accuracy, involves training SS-LCS and UCS with and without clustering-based initialization (SS-LCS$_{CI}$/SS-LCS$_{NI}$ and UCS$_{CI}$/UCS$_{NI}$, respectively), as well as their rival algorithms in a variety of real-world classification problems. The performance metric used throughout this set of experiments for algorithm comparisons is the average accuracy rate of 5 tenfold stratified cross validation runs, in line with other comparative studies in the literature (Orriols-Puig et al. 2008b; García et al. 2009).

The *second set of experiments*, aimed at investigating the "descriptive" abilities of LCS and the effect that clustering-based initialization may have on them, compares the interpretability of models produced by the nine studied learners (SS-LCS$_{NI}$, SS-LCS$_{CI}$, UCS$_{CI}$, UCS$_{NI}$, plus their five rivals). The specific focus of this set of experiments is on comparing the legibility of the knowledge representations evolved by the four candidates, in terms of the number of rules they induce, and thus, for the sake of simplicity, the "evaluate-on-training-set" method was used for each of the classification problems tackled. Again, all reported results correspond to the average of 5 runs.

Finally, regarding the learning time factor (the third requirement for the design of our experimental process), we devised a modified evaluation procedure that evaluates the model at S steps during training, where S is a user-defined parameter. This allows us to monitor the time at which each of the results is reached and arrive at qualitative conclusions on whether the CI procedure can contribute to the improvement of LCS convergence speed, and training times in general.

In order to evaluate the statistical significance of the measured differences in algorithm performance, we use the procedure suggested by Demšar (2006) for robustly comparing classifiers across multiple datasets. This procedure involves the use of the Friedman test (Friedman 1937, 1940) to establish the significance of the differences between classifier ranks and, potentially, a post-hoc test to compare classifiers with each other. In our case, the evaluation goal is twofold: (a) to compare the performance of all algorithms to each other and (b) to compare the two versions of each studied LCS algorithm, in order to validate our initial hypothesis that clustering-based initialization leads to better performance. For the first goal the Nemenyi test (Nemenyi 1963) was selected as the appropriate post-hoc test, while for the second we used the Wilcoxon signed-ranks test (Wilcoxon 1945) to examine the statistical significance of the observed performance differences between all possible pairs formed by the four studied versions of our LCS learners (SS-LCS$_{NI}$, SS-LCS$_{CI}$, UCS$_{CI}$, UCS$_{NI}$) and their rivals.

Some additional comments are in order regarding the use of the final ruleset evolved by SS-LCS and UCS (both versions). In all experiments, we employ the "fittest rule" inference strategy and a simple post-processing step, where only the subset of rules necessary to fully cover the training set is retained in the final knowledge model produced. The process used to this end is a simplified version of the one described in (Wilson 2002). First, the system discards

redundant classifiers, keeping only one copy of each rule in the population. The remaining classifiers' numerosities are, of course, increased by the sum of the numerosities of their deleted copies and their fitness-related parameters are updated accordingly (e.g. set to the values corresponding to their most experienced pre-existing copy). The resulting ruleset is evaluated on the test dataset producing the target value val_t for our performance metric of interest (the accuracy in our case). The next step involves sorting all rules in decreasing order, according to their numerosity, and the initiation of an iterative evaluation procedure that adds the (ordered) rules one by one in the final ruleset and reevaluates the latter, until the target performance value val_t is reached (or exceeded). At this point, the current ruleset is returned as the final model. It is important to note, though, that at each iteration the newly added rule is retained in the final population, only if its addition resulted in an improvement of the employed performance metric. Moreover, the process explicitly takes into account the existence of the *default rule* that unconditionally predicts the dataset's majority class when no matching classifier exists in the ruleset.

5.2 Experimental setup

5.2.1 Benchmark datasets

The benchmark datasets employed in this work are listed in Table 1 and are all readily available from the UCI repository (Asuncion and Newman 2010), except for the *web-activity* dataset that was selected from a local repository (Vavliakis et al. 2010). A major factor for choosing the particular datasets was not only their affinity to real-world domains, but also their diverse characteristics. The selected datasets, representing a wide range of problem categories, comprise a mixture of nominal and numeric attributes, a wide range of attribute numbers (4–36), classes (2–22) and imbalance ratios (1–84), several dataset sizes (101–8,124 instances), and also some cases of missing values.

Table 1 List of datasets used in the experiments

Dataset	Instances	Attributes	Classes	Imbalance
balance (bal)	625	4C	3	5.88
bupa (bupa)	345	6N	2	1.38
car (car)	1,728	6C	4	18.62
cmc (cmc)	1,473	2N/7C	3	1.89
credit-a (cre)	690	6N/9C	2	1.25
glass (gla)	214	9N	7	8.44
heart-c (h-c)	303	6N/7C	2	1.12
hepatitis (hep)	155	6N/13C	2	3.84
iris (iris)	150	4N	3	1.00
kr-vs-kp (krk)	3,196	36C	2	1.09
mushroom (mus)	8,124	22C	2	1.07
pima (pima)	768	8N	2	1.87
prim-tumor (pri)	339	17C	22	84.00
soybean (soy)	683	35C	19	11.50
tic-tac-toe (tic)	958	9C	2	1.89
voting (vot)	435	16C	2	1.59
wbcd (wbcd)	699	9N	2	1.90
web-activity (web)	741	15C	3	1.01
wine (wine)	178	13N	3	1.48
zoo (zoo)	101	1N/16C	7	10.25

Attributes can be categorical (C) or numeric (N)

5.2.2 Rival algorithms

The rival algorithms against which the studied Michigan LCS algorithms are compared are the rule-based classifiers PART, GAssist, HIDER and SIA, and the decision-tree inducing algorithm C4.5.

PART (Frank and Witten 1998) generates a decision list, using the "separate-and-conquer" approach: partial C4.5 decision trees are built in each iteration and the "best" leaf is made into a rule. Genetic Algorithms based claSSIfier sySTem (GAssist) (Bacardit, 2004) is a Pittsburgh-style LCS, initially derived from GABIL (De Jong et al. 1993). HIDER (Aguilar-Ruiz et al. 2003, 2007), an acronym for HIerarchical DEcision Rules, is an IRL approach that evolves models made available as decision lists of rules. The algorithm uses *natural coding* to represent rules and a fitness function that considers both the accuracy and the coverage of rules to guide the evolutionary search process. SIA (Venturini 1993) is another classical IRL approach that iteratively evolves rules by generalizing overly specific ones, produced based on training examples. Finally, C4.5 is the well-known decision tree induction algorithm, developed by Quinlan (1993), that builds decision trees from a set of training data using the concept of Information Entropy.

For PART and C4.5 we employed their WEKA implementations (Witten and Frank 2005), for GAssist, HIDER and SIA the implementations provided by the ML tool Keel (Alcalá-Fdez et al. 2009), while for SS-LCS and UCS we used our own implementations codified in Java.[3]

5.2.3 Parameter setup

The parameters used through all experiments (except for the datasets reported in Table 2) for SS-LCS and UCS are: $|\mathbf{P}| = 1{,}000$, $P_{\#} = 0.33$, $P^{ci}_{\#A} = P^{ci}_{\#V} = 0.33$, $\theta_{\mathrm{GA}} = 50$, $\chi = 0.8$, $\mu = 0.04$, $GASubsumption = true$, $\theta_{\mathrm{sub}} =$

[3] Code for both algorithms, as well as the clustering-based initialization process, is available upon request from the first author.

Table 2 LCS parameters for the datasets with a high imbalance ratio

Dataset	$\|\mathbf{P}\|$	θ_{GA}
balance	1,000	150
car	2,000	300
glass	2,000	150
primary-tumor	6,400	300
soybean	2,000	150

50, $\theta_{del} = 50$, $\tau_s = 0.4 \cdot |C|$, $\tau_d = 0.25 \cdot |P|$, $\nu = 10$, $p = 10, R = 1$, $\text{tpr}_{min} = 1 - 10^{-4}$, $acc_0 = 0.999$, $\beta = 0.2$ and 100,000 learning iterations.

For the datasets in Table 2 that exhibit a high imbalance ratio (i.e., a high ratio between the dataset's prevalent and minority class's number of instances) a modest tuning process was employed, resulting in a slight deviation from the general parameter setup of Sect. 5.2. This tuning is actually part of the algorithms' implementation (i.e., it is performed automatically, if necessary, with no user intervention) and is mainly guided by the dataset's imbalance ratio. More specifically, depending on the dataset's characteristics, one or both of two parameters may be automatically changed: the allowed number of microclassifiers ($|\mathbf{P}|$), and the GA invocation rate (θ_{GA}). The GA invocation rate is approximately selected according to the bound defined by $\theta_{GA} = k \cdot ir$, where ir is the imbalance ratio of the dataset and k is an arbitrary constant, defining the number of updates of a classifier belonging to a starved niche before it receives a genetic event (Orriols-Puig and Bernadó-Mansilla, 2008a). On the other hand, the number of microclassifiers is configured according to the bound for the minimum population required to guarantee a sufficient initial supply of rules $|\mathbf{P}| = O[n \cdot (1 + ir)]$, where n is the number of classes and ir is the imbalance ratio of the dataset (Orriols-Puig et al. 2007).

5.3 Comparative analysis of results

5.3.1 1st set of experiments: predictive accuracy

Tables 3 and 4 summarize the results of the first set of experiments, where five intermediate evaluation steps were used. The average achieved accuracy rate (over 5 tenfold cross-validation runs) is reported for each evaluation step for SS-LCS (Table 3) and UCS (Table 4) with and without the clustering-based initialization component. The best result per dataset and algorithm is marked in bold. The last two columns summarize our observations regarding the competence of the CI procedure in providing better accuracy results.

We can easily observe that our initial hypothesis that the CI procedure can lead to better final solutions is confirmed for both studied LCS algorithms. SS-LCS$_{CI}$ outperforms SS-LCS$_{NI}$ in 15 out of the 20 studied cases (marked with the ✓ symbol in the "Result.acc" column). From the rest of the cases, 1 corresponds to a tie between the two versions of the algorithm (marked with ≡) and 4 to failures of the CI component to achieve a better solution (marked with ✗). It is interesting to note, though, that in 4 of the 5 tie/failure cases, SS-LCS$_{CI}$ still manages to reach the best solution faster (at an earlier evaluation step). The situation is similar for UCS, with UCS$_{CI}$ outperforming UCS$_{NI}$ in 15 out of the 20 studied cases and failing to achieve a better solution in 5.

Regarding the time required to achieve the best solution, we observe a clear tendency of the CI versions to outperform their rivals. Compared with SS-LCS$_{NI}$, SS-LCS$_{CI}$ arrives at the best solution at an earlier evaluation step 11 out of the 20 times (marked with the ▲ symbol in the "Result.step" column) and later in 4 cases (marked with ∇), while there are also 5 ties (marked with ≡). For UCS, on the other hand, the CI version arrives at the best solution at an earlier evaluation step 12 out of the 20 times and later in 3, while there are also 5 ties with UCS$_{NI}$. Overall, results indicate a significant improvement of training times when optimizing the achieved accuracy is our primary goal, with SS-LCS$_{CI}$ arriving at the best solution at the 2.90th evaluation step on average—that is with a 24.14% improvement over SS-LCS$_{NI}$, which arrives at the best solution at the 3.60th step—and UCS$_{CI}$ improving the average step of its rival by 28.3% (bringing it down to 2.65 steps from 3.40).

Table 5 reports the results of the comparison of the studied LCS algorithms (with and without the CI component) with their rival ML techniques—namely C4.5, PART, GAssist, HIDER and SIA—by summarizing their accuracy rates on all datasets used in this study. Along with the average accuracy rates, we also report the corresponding standard deviations per dataset (over the 5 tenfold cross-validation runs), as well as each algorithm's overall average rank (row labeled "Rank") and its position in the final ranking (row labeled "Pos").

Based on the measured accuracy results, the average rank provides a clear indication of the studied algorithms relative performance: SS-LCS$_{CI}$ ranks first and UCS$_{CI}$ third, both clearly outperforming their non-CI versions, ranking second (SS-LCS$_{NI}$), and seventh (UCS$_{NI}$), respectively. From the rival algorithms, only C4.5, PART and GASSSIT manage to outrank at least one version of the studied LCS algorithms, namely the relatively poor performing UCS$_{NI}$.

Regarding the statistical significance of the measured differences in algorithm ranks, the use of the Friedman test rejects the null hypothesis (at $\alpha = 0.01$) that all algorithms perform equivalently, and the Nemenyi post-hoc test detects significant differences between SS-LCS$_{CI}$ and each

Table 3 Classification accuracy for SS-LCS—with (CI) and without (NI) the clustering-based initialization component—for the five intermediate evaluation steps

	$SS\text{-}LCS_{NI}$ evaluation steps					$SS\text{-}LCS_{CI}$ evaluation steps					Result	
Dataset	1	2	3	4	5	1	2	3	4	5	acc	step
bal	83.65	84.54	**85.06**	84.19	84.19	84.83	**85.41**	84.64	84.83	84.38	✓	▲
bupa	**68.41**	66.73	66.55	64.41	65.68	66.73	**68.81**	66.21	66.21	64.41	✓	∇
car	89.97	91.91	92.86	92.99	**93.44**	89.14	90.88	92.04	92.80	**93.47**	✓	≡
cmc	46.14	45.83	46.57	**46.60**	45.44	**46.75**	46.73	46.72	46.30	46.50	✓	▲
cre	86.03	86.00	86.38	86.09	**87.16**	86.03	86.03	**86.67**	85.80	85.57	✗	▲
gla	71.03	70.10	**71.31**	71.31	70.84	**71.68**	70.65	70.93	71.22	71.22	✓	▲
h-c	**80.33**	79.14	78.88	78.15	78.75	**81.85**	80.66	81.39	80.20	80.66	✓	≡
hep	83.61	**84.13**	82.58	81.55	81.55	82.07	**82.97**	82.71	82.45	82.71	✗	≡
iris	95.20	95.33	95.20	95.20	**95.60**	95.87	**96.00**	95.87	95.73	95.87	✓	▲
krk	94.99	96.11	97.03	97.30	**97.59**	95.21	96.45	97.25	97.54	**97.73**	✓	≡
mus	98.68	99.14	99.33	99.42	**99.59**	98.81	99.19	99.44	99.58	**99.62**	✓	≡
pima	**75.18**	74.22	74.58	74.74	74.74	74.58	74.77	74.09	74.61	**75.73**	✓	∇
pri	40.24	40.89	41.95	41.71	**42.30**	40.65	**42.30**	41.12	41.48	41.18	✓	▲
soy	87.96	89.58	91.07	91.07	**92.33**	88.99	90.95	92.03	**92.33**	92.27	≡	▲
tic	99.00	99.27	99.44	**99.56**	99.44	98.83	**99.44**	99.39	99.41	99.42	✓	▲
vot	94.85	94.80	94.89	94.66	**94.99**	94.80	95.26	**95.31**	95.31	94.48	✗	▲
wbcd	92.82	94.14	94.19	**94.77**	94.62	92.73	93.96	94.33	94.85	**94.96**	✓	∇
web	63.78	63.62	64.35	**64.48**	64.48	64.34	63.75	**65.61**	63.18	63.51	✓	▲
wine	95.62	95.62	95.73	**96.07**	95.96	94.72	95.39	**95.96**	95.62	95.62	✗	▲
zoo	**95.84**	95.45	95.64	95.84	95.05	96.24	**96.63**	96.63	96.24	95.84	✓	∇

The reported values are averages over 5 tenfold cross-validation runs, while the best achieved result per dataset and algorithm is shown in bold. The last two columns report the result of the overall comparison, in terms of classification accuracy and the time to achieve the best solution. A ✓(▲) sign indicates that the CI version achieves better classification accuracy (faster convergence) than the NI one. A ✗ (∇) sign indicates that the CI version achieves worse classification accuracy (slower convergence), while a ≡ sign represents a tie between the two versions of the algorithm, in terms of the property studied in the column

of the two IRL rival methods (HIDER and SIA) at $\alpha = 0.05$. $SS\text{-}LCS_{NI}$ also performs significantly better than HIDER, at the same confidence level. No significant difference is, though, detected between the two versions (with and without CI) of SS-LCS or UCS.

However, the Wilcoxon signed-ranks test is more powerful in pairwise comparisons and also allows us to compute the confidence level at which the difference between two algorithms may be considered significant. These confidence levels are reported for $SS\text{-}LCS_{CI}$, $SS\text{-}LCS_{NI}$, UCS_{CI}, and UCS_{NI} in the last four rows of Table 5, unless smaller than 80%. A careful examination of the results allows us to conclude that $SS\text{-}LCS_{CI}$ performs better than all its rivals at a level of confidence greater than 80%, including $SS\text{-}LCS_{NI}$ at 96.53%. Moreover, although the pairwise comparisons yield fewer statistically significant results for UCS_{CI}, the latter significantly outperforms UCS_{NI} and both IRL methods at a level of confidence greater than 90%—notice the confidence levels noted by a plus (+) sign in row "WxUCI".

5.3.2 2nd set of experiments: readability of evolved models

Table 6 summarizes the results (again averaged over 5 runs) of the second set of experiments, reporting both the number of rules produced per algorithm-dataset pair

Table 4 Classification accuracy for UCS—with (CI) and without (NI) the clustering-based initialization component—for the five intermediate evaluation steps

	UCS_{NI} evaluation steps					UCS_{CI} evaluation steps					Result	
Dataset	1	2	3	4	5	1	2	3	4	5	acc	step
bal	68.38	67.97	74.62	76.74	**78.78**	67.58	75.81	79.36	**80.16**	80.03	✓	▲
bupa	65.05	64.64	62.61	63.89	**66.38**	62.67	63.31	64.58	63.13	**66.67**	✓	≡
car	84.67	87.94	90.93	92.95	**94.69**	85.90	89.99	91.73	93.79	**95.05**	✓	≡
cmc	48.95	46.46	47.14	48.06	**49.04**	**50.77**	48.15	48.43	48.28	49.26	✓	▲
cre	81.94	**82.67**	82.20	81.94	81.88	**83.10**	82.75	82.35	82.70	81.80	✓	▲
gla	61.21	67.66	66.64	67.76	**68.97**	65.70	69.72	**71.78**	70.65	71.40	✓	▲
h-c	**77.96**	76.90	76.44	76.11	75.71	**79.54**	79.01	79.47	79.34	79.47	✓	≡
hep	80.65	**80.77**	79.74	79.48	79.48	**80.90**	80.00	79.48	78.19	78.71	✓	▲
iris	93.07	94.40	94.93	94.80	**95.33**	**95.60**	95.07	95.47	94.93	95.46	✓	▲
krk	97.15	98.57	98.86	98.93	**99.13**	97.87	98.45	98.99	99.06	**99.20**	✓	≡
mus	99.98	99.99	99.99	**100.00**	99.99	99.95	99.98	**100.00**	99.99	99.99	✓	▲
pima	72.19	73.05	72.21	**73.75**	71.54	71.72	70.29	71.07	71.51	**72.73**	✗	∇
pri	35.69	**39.35**	38.58	39.06	37.05	37.34	38.94	37.76	**40.12**	37.88	✓	∇
soy	87.26	89.34	91.21	91.30	**91.71**	86.73	89.75	90.86	**91.83**	91.57	✓	▲
tic	99.19	**99.56**	99.41	99.21	98.98	98.44	**99.37**	99.35	99.12	99.10	✗	≡
vot	94.16	**94.89**	94.53	94.89	94.89	93.97	94.43	94.16	**94.66**	94.66	✗	∇
wbcd	95.88	**96.17**	95.68	95.31	95.14	**96.19**	95.91	95.57	95.74	95.45	✓	▲
web	65.59	**66.43**	65.48	64.99	64.16	**65.78**	64.67	64.08	65.35	65.42	✗	▲
wine	94.38	95.06	**95.62**	94.94	94.72	**95.73**	95.17	94.38	93.82	94.16	✓	▲
zoo	95.84	**96.44**	95.64	95.84	96.44	**95.45**	95.05	94.85	95.25	94.85	✗	▲

The reported values are averages over 5 tenfold cross-validation runs, while the best achieved result per dataset and algorithm is shown in bold. The last two columns report the result of the overall comparison, in terms of classification accuracy and the time to achieve the best solution. A ✓(▲) sign indicates that the CI version achieves better classification accuracy (faster convergence) than the NI one. A ✗ (∇) sign indicates that the CI version achieves worse classification accuracy (slower convergence), while a ≡ sign represents a tie between the two versions of the algorithm, in terms of the property studied in the column

("NR" columns) and the classification accuracy achieved ("acc" columns), to ensure that the model sizes are evaluated at comparable performance levels, in terms of predictive accuracy. For this set of experiments, the five intermediate evaluation steps used for the two versions of the studied LCS algorithms (SS-LCS and UCS) are not all reported for brevity. Instead, only the step at which the corresponding ruleset size was obtained is shown in the columns labeled "Step". The symbols depicted in the "Result" columns summarize the result of the overall comparison of the CI versions of the algorithms with their non-CI rivals: the signs in column "Result.NR" refer to the comparison based on the number of rules, while the ones in column "Result.Step" refer to the learning iterations required to achieve the best solution. The interpretation of the symbols is the same as in the first set of experiments, with a ✓(▲) sign indicating that the CI version evolves more compact models (achieves faster convergence) than the corresponding NI method, a ✗(∇) sign indicating the opposite case, and ≡ being the symbol used for ties.

Inspecting the reported results, one may easily observe that there are 16 cases where SS-LCS_{CI} produces fewer rules than SS-LCS_{NI}, 3 cases where SS-LCS_{CI} produces more rules and 1 tie. For UCS, the CI version outperforms the non-CI one in 14 cases, while there are also 5 cases where UCS_{CI} produces more rules than UCS_{NI} and 1 tie. Overall, given that

Table 5 Average classification accuracy per algorithm over 5 tenfold cross-validation runs

Dataset	C4.5	PART	GAssist	HIDER	SIA	SS-LCS$_{NI}$	SS-LCS$_{CI}$	UCS$_{NI}$	UCS$_{CI}$
bal	76.64 ± 4.33	83.52 ± 5.65	79.76 ± 0.52	71.56 ± 0.35	82.45 ± 0.58	85.06 ± 0.52	**85.41 ± 0.38**	78.78 ± 4.90	80.16 ± 5.34
bup	68.70 ± 8.74	63.81 ± 6.36	64.24 ± 2.23	63.31 ± 2.57	62.40 ± 3.18	68.41 ± 1.47	**68.81 ± 1.58**	66.38 ± 1.40	66.67 ± 1.62
car	92.36 ± 2.10	**95.77 ± 1.45**	90.98 ± 0.86	70.02 ± 0.00	93.21 ± 0.43	93.44 ± 1.38	93.47 ± 1.71	94.69 ± 4.00	95.05 ± 3.58
cmc	52.13 ± 3.60	49.14 ± 4.13	**54.15 ± 1.20**	51.61 ± 0.44	48.16 ± 0.87	46.60 ± 0.49	46.75 ± 0.20	49.04 ± 1.13	50.77 ± 1.09
cre	86.09 ± 3.75	85.36 ± 4.90	84.67 ± 1.08	82.77 ± 0.98	67.01 ± 0.88	**87.16 ± 0.49**	86.67 ± 0.41	82.67 ± 0.33	83.10 ± 0.49
gla	66.82 ± 7.94	67.58 ± 7.21	63.13 ± 1.00	64.47 ± 0.80	71.15 ± 0.90	71.31 ± 0.50	71.68 ± 0.38	68.97 ± 3.04	**71.78 ± 2.45**
h-c	77.85 ± 7.94	81.84 ± 6.62	79.11 ± 1.06	75.11 ± 1.59	66.25 ± 2.20	80.33 ± 0.80	**81.85 ± 0.66**	77.96 ± 0.86	79.54 ± 0.21
hep	83.79 ± 7.24	84.46 ± 7.59	**88.82 ± 1.36**	84.86 ± 1.14	79.15 ± 1.71	84.13 ± 1.18	82.97 ± 0.34	80.77 ± 0.64	80.90 ± 1.07
iris	96.00 ± 5.62	94.00 ± 5.84	**96.80 ± 0.73**	95.33 ± 0.47	95.20 ± 1.66	95.60 ± 0.17	96.00 ± 0.09	95.33 ± 0.87	95.60 ± 0.29
krk	**99.44 ± 0.48**	99.06 ± 0.59	96.97 ± 0.50	94.33 ± 0.01	99.32 ± 0.10	97.59 ± 1.06	97.73 ± 1.03	99.13 ± 0.80	99.20 ± 0.55
mus	**100.0 ± 0.00**	**100.0 ± 0.00**	99.52 ± 0.17	98.40 ± 0.31	99.98 ± 0.02	99.59 ± 0.35	99.62 ± 0.33	**100.0 ± 0.01**	**100.0 ± 0.02**
pima	73.83 ± 5.66	75.27 ± 3.93	74.76 ± 0.97	73.56 ± 0.31	71.20 ± 0.81	75.18 ± 0.35	**75.73 ± 0.60**	73.75 ± 0.86	72.73 ± 0.90
pri	39.80 ± 5.04	40.70 ± 4.64	**45.60 ± 0.46**	33.99 ± 1.74	30.49 ± 0.62	42.30 ± 0.84	42.30 ± 0.61	39.35 ± 1.54	40.12 ± 1.12
soy	91.51 ± 2.47	91.94 ± 2.33	69.58 ± 1.69	88.77 ± 0.53	90.19 ± 0.69	**92.33 ± 1.68**	**92.33 ± 1.41**	91.71 ± 1.86	91.83 ± 2.07
tic	85.07 ± 4.49	94.47 ± 3.15	95.76 ± 0.52	65.34 ± 0.00	**99.73 ± 0.16**	99.56 ± 0.22	99.44 ± 0.26	99.56 ± 0.22	99.37 ± 0.38
vot	96.33 ± 3.42	94.71 ± 3.58	**96.97 ± 0.55**	96.95 ± 0.00	92.85 ± 1.85	94.99 ± 0.12	95.31 ± 0.38	94.89 ± 0.33	94.66 ± 0.31
wbcd	94.56 ± 3.63	93.85 ± 2.94	95.22 ± 0.45	**96.36 ± 0.25**	95.57 ± 0.25	94.77 ± 0.77	94.96 ± 0.90	96.17 ± 0.42	96.19 ± 0.29
web	65.86 ± 3.61	67.34 ± 4.65	**67.83 ± 1.09**	61.86 ± 1.26	63.76 ± 1.37	64.48 ± 0.41	65.61 ± 0.96	66.43 ± 0.83	65.78 ± 0.68
wine	93.86 ± 5.52	93.27 ± 5.80	93.22 ± 1.12	77.42 ± 2.44	94.93 ± 0.56	**96.07 ± 0.21**	95.96 ± 0.46	95.62 ± 0.46	95.73 ± 0.78
zoo	92.18 ± 8.94	92.18 ± 8.94	92.58 ± 1.14	94.55 ± 1.79	93.16 ± 1.40	95.84 ± 0.33	**96.63 ± 0.33**	96.44 ± 0.37	95.45 ± 0.26
Rank	5.05	4.83	5.00	6.80	6.50	4.05	3.35	5.08	4.35
Pos.	6	4	5	9	8	2	1	7	3
WxSNI	89.95^{+}	84.40^{+}		99.68^{+}	99.35^{+}		96.53^{-}	85.25^{+}	
WxSCI	94.65^{+}	91.40^{+}	84.40^{+}	99.75^{+}	99.64^{+}	96.53^{+}		92.68^{+}	89.15^{+}
WxUNI				99.34^{+}	99.10^{+}	85.25^{-}	92.68^{-}		90.10^{-}
WxUCI				99.48^{+}	99.85^{+}		89.15^{-}	90.10^{+}	

The average ranks used in the computation of the Friedman test are reported in the row labeled "Rank", while row "Pos" holds each algorithm's position in the overall ranking. The last four rows report the confidence level of the Wilcoxon signed-ranks test for SS-LCS$_{CI}$ (WxSCI), SS-LCS$_{NI}$ (WxSNI), UCS$_{CI}$ (WxUCI) and UCS$_{NI}$ (WxUNI) with respect to the method in the column (unless smaller than 80%)

the CI approaches also clearly outperform their rivals in terms of prediction accuracy (see Table 5), we consider these results encouraging and indicative of LCS potential to evolve tractable, yet effective, models in supervised classification tasks, when equipped with the clustering-based initialization component.

Regarding the time required to achieve the best solution (the most compact ruleset), results in this case are indicative of a less significant improvement, compared with the first set of experiments. SS-LCS$_{CI}$ arrives at the best solution at an earlier evaluation step than SS-LCS$_{NI}$ in 8 out of the 20 cases and later in 4, while there are also 8 ties. UCS$_{CI}$, on the other hand, achieves 11 wins, 4 losses, and 5 ties, with regard to UCS$_{NI}$. Overall, the improvement of training times, when optimizing the readability of evolved rulesets is our primary goal, is approximately 10%, with SS-LCS$_{CI}$ arriving at the best solution at the 2.5th evaluation step on average—compared with 2.75 steps for SS-LCS$_{NI}$—and UCS$_{CI}$ improving its non-CI version by 0.35 steps ($\approx$7,000 learning iterations), that is arriving at the best solution at the 3.15th evaluation step on average.

Table 7 reports the results of the comparison of the studied LCS algorithms (with and without the CI component) with their rival ML techniques, by summarizing the final ruleset sizes on all datasets used in this study. Along with the average number of rules, we also report the corresponding achieved accuracy rates per dataset (averaged over 5 runs), as well as each algorithm's overall average rank (row labeled "Rank") and its position in the final ranking (row labeled "Pos").

Based on the reported results and the corresponding average ranks, one may easily draw some initial conclusions regarding the studied algorithms' relative performance: while SS-LCS$_{CI}$ and UCS$_{CI}$ are outranked by HIDER and GAssist, they still achieve a good overall ranking (fourth and third, respectively), clearly outperforming their non-CI versions, ranking seventh (SS-LCS$_{NI}$) and fifth (UCS$_{NI}$), respectively. Especially regarding HIDER, it is also worth noting that while it outperforms both SS-LCS$_{CI}$ and UCS$_{CI}$ in terms of the number of rules, it presents particularly low predictive accuracy (in the experiments reported in Table 7), ranking last when

Table 6 Size of the models (number of rules) evolved by the two versions of the studied LCS algorithms—with (CI) and without (NI) the clustering-based initialization component

Dataset	$SS\text{-}LCS_{NI}$			$SS\text{-}LCS_{CI}$			Result		UCS_{NI}			UCS_{CI}			Result	
	acc	NR	Step	acc	NR	Step	NR	Step	acc	NR	Step	acc	NR	Step	NR	Step
bal	87.04	20.8	1	87.55	23.2	1	✗	≡	76.35	5.8	2	76.86	5.2	1	✓	▲
bupa	86.09	32.2	4	86.15	31.8	3	✓	▲	78.32	17.4	5	81.51	18.8	5	✗	≡
car	97.37	50.0	5	96.57	44.6	4	✓	▲	85.85	19.8	2	81.18	8.0	1	✓	▲
cmc	56.52	43.4	1	58.18	54.6	5	✗	∇	49.15	12.0	2	45.77	7.8	1	✓	▲
cre	90.98	22.0	3	90.35	19.6	2	✓	▲	88.43	20.0	3	90.15	21.6	3	✗	≡
gla	95.05	33.4	5	94.67	27.2	5	✓	≡	87.48	28.8	1	93.93	27.4	4	✓	∇
h-c	91.35	17.4	1	90.76	13.8	1	✓	≡	93.66	14.0	5	93.66	13.8	4	✓	▲
hep	92.78	6.2	1	92.39	5.2	1	✓	≡	96.00	7.6	4	95.74	6.6	5	✓	∇
iris	96.80	5.6	1	96.67	6.6	1	✗	≡	96.00	5.8	1	98.13	8.6	3	✗	∇
krk	97.36	8.8	2	96.96	8.0	1	✓	▲	99.63	23.2	5	99.32	21.8	4	✓	▲
mus	98.82	4.2	1	99.12	3.6	4	✓	∇	100.00	7.2	4	100.00	6.6	4	✓	≡
pima	84.03	41.2	5	83.80	38.6	1	✓	▲	75.29	16.6	5	74.74	20.6	3	✗	▲
pri	61.65	65.4	5	62.18	56.8	2	✓	▲	62.01	39.4	4	61.59	35.4	3	✓	▲
soy	96.34	50.2	5	96.37	45.4	5	✓	≡	96.37	47.2	5	96.81	42.0	4	✓	▲
tic	97.95	8.0	1	98.33	8.0	2	≡	∇	98.33	8.2	3	98.33	8.6	4	✗	∇
vot	96.00	4.4	1	96.50	3.8	1	✓	≡	94.94	3.6	2	95.08	2.6	1	✓	▲
wbcd	95.91	6.8	1	95.62	6.2	1	✓	≡	98.46	14.2	5	98.57	11.4	5	✓	≡
web	83.97	91.0	4	84.89	88.8	1	✓	▲	77.33	52.0	3	77.97	50.2	1	✓	▲
wine	99.10	7.0	5	98.88	6.0	4	✓	▲	99.33	7.2	5	99.55	6.6	5	✓	≡
zoo	99.21	9.0	3	99.60	8.6	5	✓	∇	99.60	8.6	4	99.01	8.6	2	≡	▲

The columns labeled "Result" report the result of the overall comparison, in terms of the number of rules ("NR") and the time to achieve the best solution ("Step"). A ✓(▲) sign indicates that the CI version produces more compact models (achieves faster convergence) than the NI one. A ✗ (∇) sign indicates that the CI version produces less compact models (achieves slower convergence), while a ≡ sign represents a tie between the two versions of the algorithm, in terms of the property studied in the column

evaluated based on accuracy and, more specifically, achieving less accurate models than $SS\text{-}LCS_{CI}$ and UCS_{CI} in more than half of the studied problems. This observation is also true, to a lesser extent, for GAssist that ranks eighth when evaluated based on accuracy and achieves less accurate models than $SS\text{-}LCS_{CI}$ and UCS_{CI} in 7 out of the 20 studied problems. Overall, given that the studied LCS use a very simple ruleset reduction technique and, more importantly, that they clearly outperform their rivals in terms of prediction accuracy (see Table 5), we consider these results as further indications of the potential of the CI component for boosting the performance of LCS in supervised classification tasks.

Regarding the statistical significance of the measured differences in algorithm ranks, the use of the Friedman test rejects the null hypothesis (at $\alpha = 0.01$) that all algorithms perform equivalently, and the Nemenyi post-hoc test does not detect any significant differences between the two top performing algorithms (GAssist and HIDER) and the CI versions of the studied LCS algorithms. However, GAssist significantly outperforms $SS\text{-}LCS_{NI}$ at $\alpha = 0.05$ and UCS_{NI} at $\alpha = 0.1$, while HIDER significantly outperforms $SS\text{-}LCS_{NI}$ at $\alpha = 0.05$. Moreover, the use of the Nemenyi test yields no significant differences between the two versions (with and without CI) of SS-LCS or UCS. The Wilcoxon signed-ranks test, however, allows us to perform a more focused pairwise comparison of our target algorithms and reveals that for both studied LCS algorithms, namely SS-LCS and UCS, the clustering-based initialization component significantly improves the baseline

Table 7 Average size of models (number of rules—column "NR") per algorithm over five runs

Dataset	C4.5		PART		GAssist		HIDER		SIA		SS-LCS$_{NI}$		SS-LCS$_{CI}$		UCS$_{NI}$		UCS$_{CI}$	
	acc	NR	acc	NR	acc	NR	acc	NR	acc	NR	acc	NR	acc	NR	acc	NR	acc	NR
bal	90.08	52	95.04	47	84.77	8.6	**75.20**	**4.0**	88.80	118.8	87.04	20.8	87.55	23.2	76.35	5.8	76.86	5.2
bupa	84.64	26	86.09	15	**79.30**	**7.0**	71.88	8.8	100.0	236.0	86.09	32.2	86.15	31.8	78.32	17.4	81.51	18.8
car	96.30	131	98.67	68	93.48	14.8	**70.02**	**1.0**	97.73	242.4	97.37	50.0	96.57	44.6	85.85	19.8	81.18	8.0
cmc	71.15	157	75.70	191	**57.90**	**5.2**	54.99	13.8	91.28	685.2	56.52	43.4	58.18	54.6	49.15	12.0	45.77	7.8
cre	90.73	30	93.48	23	**90.17**	**5.6**	89.77	27.2	100.0	649.8	90.98	22.0	90.35	19.6	88.43	20.0	90.15	21.6
gla	93.93	26	91.59	13	**73.27**	**5.0**	90.37	28.0	100.0	209.8	95.05	33.4	94.67	27.2	87.48	28.8	93.93	27.4
h-c	92.08	30	93.40	25	**91.49**	**6.8**	84.46	12.2	100.0	272.8	91.35	17.4	90.76	13.8	93.66	14.0	93.66	13.8
hep	92.26	11	95.48	8	98.32	5.4	97.75	6.2	98.32	77.6	92.78	6.2	**92.39**	**5.2**	96.00	7.6	95.74	6.6
iris	98.00	5	**97.33**	**3**	98.27	4.0	97.33	3.2	100.0	16.4	96.80	5.6	96.67	6.6	96.00	5.8	98.13	8.6
krk	99.66	31	99.75	23	97.07	6.6	**94.37**	**3.0**	99.95	80.0	97.36	8.8	96.96	8.0	99.63	23.2	99.32	21.8
mus	100.0	25	100.0	13	99.80	4.6	98.17	6.8	99.94	25.4	98.82	4.2	**99.12**	**3.6**	100.0	7.2	100.0	6.6
pima	84.12	20	81.25	13	**81.59**	**7.4**	78.57	19.0	100.0	727.8	84.03	41.2	83.80	38.6	75.29	16.6	74.74	20.6
pri	61.36	47	61.36	43	**48.50**	**7.8**	81.21	35.8	68.02	166.0	61.65	65.4	62.18	56.8	62.01	39.4	61.59	35.4
soy	96.34	61	96.34	40	**69.60**	**18.4**	95.09	30.0	99.00	89.6	96.34	50.2	96.37	45.4	96.37	47.2	96.81	42.0
tic	93.74	95	96.35	50	97.12	19.0	**65.34**	**1.0**	100.0	34.4	97.95	8.0	98.33	8.0	98.33	8.2	98.33	8.6
vot	97.24	6	97.47	7	98.99	5.8	**96.98**	**2.0**	96.87	20.0	96.00	4.4	96.50	3.8	94.94	3.6	95.08	2.6
wbcd	98.14	14	98.43	10	98.74	5.2	**97.22**	**2.2**	99.86	45.4	95.91	6.8	95.62	6.2	98.46	14.2	98.57	11.4
web	79.35	54	84.89	54	75.76	11.4	**75.99**	**10.0**	97.62	260.2	83.97	91.0	84.89	88.8	77.33	52.0	77.97	50.2
wine	98.88	5	98.88	5	**99.21**	**4.2**	99.66	35.6	100.0	178.0	99.10	7.0	98.88	6.0	99.33	7.2	99.55	6.6
zoo	99.01	9	99.01	8	98.02	7.2	**99.60**	**7.0**	100.0	10.2	99.21	9.0	99.60	8.6	99.60	8.6	99.01	8.6
Rank	6.68		5.15		2.20		2.83		8.90		5.68		4.50		4.90		4.18	
Pos	8		6		1		2		9		7		4		5		3	

The "acc" columns also report the achieved accuracy rates for the corresponding models. The average ranks used in the computation of the Friedman test are reported in the row labeled "Rank", while row "Pos" holds each algorithm's position in the overall ranking

approach at a confidence level greater than 90%. More specifically, SS-LCS$_{CI}$ performs better than SS-LCS$_{NI}$ at a 97.7% level of confidence, while UCS$_{CI}$ significantly outperforms UCS$_{NI}$, with 91.65% confidence.

6 Further study of the CI component's design choices

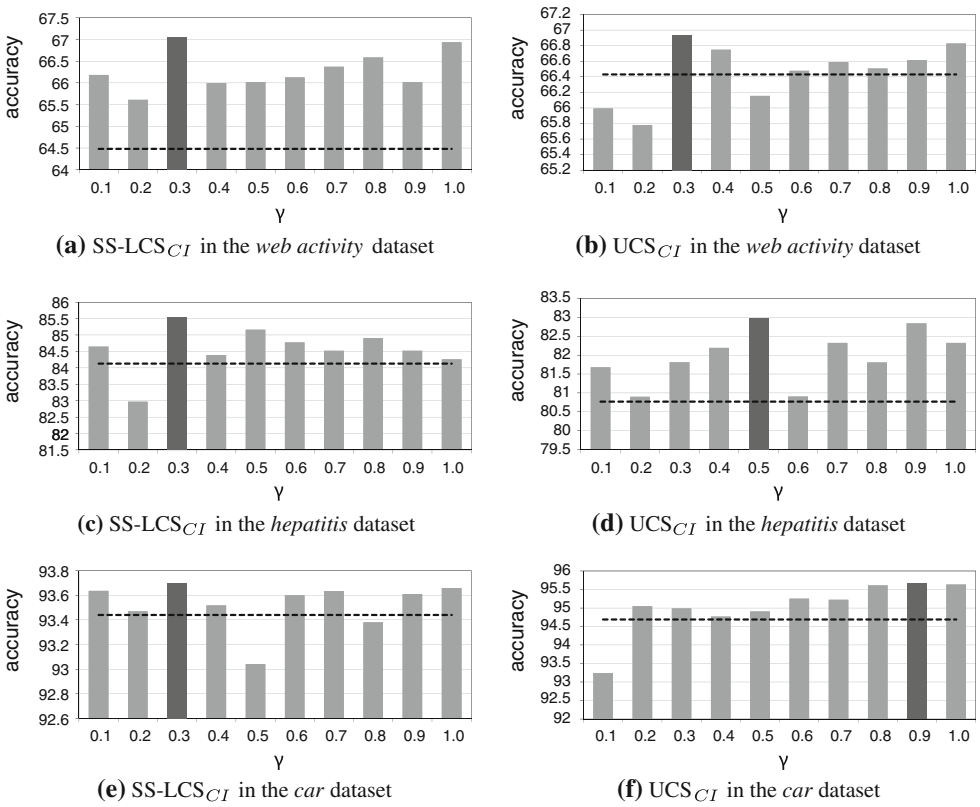
With an interest in further assessing the potential of the proposed CI method for improving the performance of supervised LCS, we performed two additional sets of experiments. Those experiments, although less extensive than those presented in Sect. 5, aim at evaluating the variability of results (achieved with the CI method) that is due to two important design choices:

1. the number of clusters based on which the initial ruleset is created (i.e., the choice of a specific value for the γ parameter); and
2. the distribution of initial rules, with respect to the distribution of instances in the training dataset.

Regarding the second point, our initial approach was to create initial rules (based on clustering) proportionally to each class' prevalence in the training dataset. For example, for a two-class problem with 100 instances of class A and 50 instances of class B, $\gamma \cdot 100$ rules would be created by clustering instances of class A and $\gamma \cdot 50$ rules by clustering instances of class B. Thus, the initial population of rules would be biased towards rules advocating the majority class A.

Among the various possible strategies for "tuning" the distribution of initial rules, in this Section, we provide an initial investigation of the "uniform-distribution" strategy—indicated by appending the subscript *CIu* to the algorithm's name in what follows. The latter strategy creates the same number of initial rules per class, essentially weighing the γ parameter by the inverse of each class' relative prevalence with respect to the minority class. For our previous example, thus, the "uniform-distribution" CI strategy would yield $\gamma \cdot 50$ clustering-based initial rules per class.

Based on the above discussion, the two sets of presented experiments involve training the CI versions of SS-LCS and UCS with various values of the γ parameter, raining from 0.1 to 1 (with 0.1 steps). For the first set of experiments, the CI component corresponds exactly to the



(a) SS-LCS$_{CI}$ in the *web activity* dataset

(b) UCS$_{CI}$ in the *web activity* dataset

(c) SS-LCS$_{CI}$ in the *hepatitis* dataset

(d) UCS$_{CI}$ in the *hepatitis* dataset

(e) SS-LCS$_{CI}$ in the *car* dataset

(f) UCS$_{CI}$ in the *car* dataset

Fig. 2 Average classification accuracy (over 5 tenfold cross-validation runs) for SS-LCS$_{\text{CI}}$ and UCS$_{\text{CI}}$. For each problem-algorithm pair, results for various values of the γ parameter (x-axis) are reported. The *dotted line* marks the accuracy of the baseline approach (without the CI component) for each problem, as measured in the experiments of Table 5

description provided in Sect. 4, while for the second set the default distribution strategy is replaced by the "uniform-distribution" one, thus removing any potential bias of the initial clustering-based population towards prevalent classes.

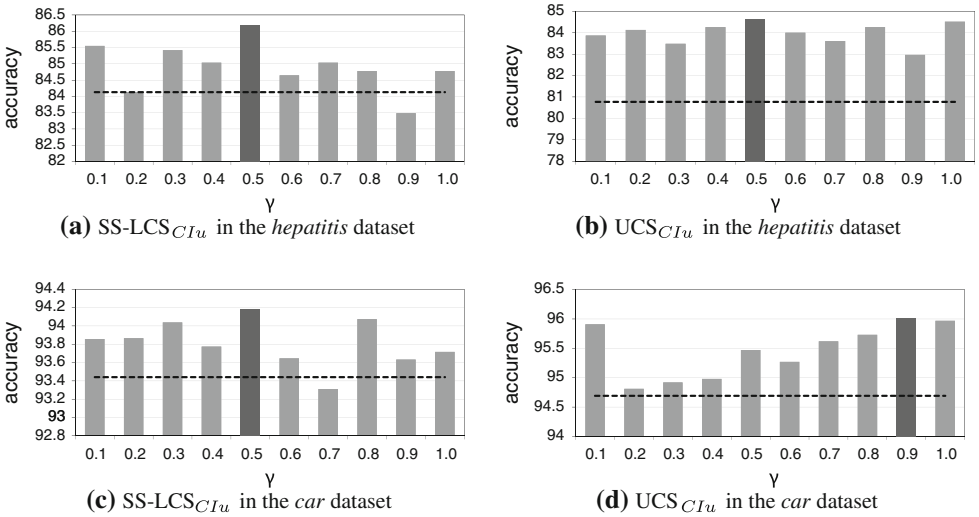
Given that both sets of experiments are meant as initial investigations of the CI component's sensitivity to parameter and/or design choices, we have only focused on a small subset of the 20 problems introduced in Sect. 5: (i) the *web activity* dataset that is a relatively complex dataset (in terms of the number of instances and attributes) with no imbalance; (ii) the *hepatitis* dataset that, while relatively small, has a medium imbalance rate (of 3.84) and presents difficulties to all versions of the studied LCS algorithms; and (iii) the *car* dataset that has the second highest imbalance rate and a relatively large number of training instances. Finally, it is also worth noting that, for each distinct combination of dataset-algorithm pair and γ value, the reported results are averaged over 5 tenfold cross-validation runs.

Figure 2 depicts the classification accuracy achieved by SS-LCS$_{\text{CI}}$ and UCS$_{\text{CI}}$ for each of the studied problems and the various values of the γ parameter (along the X-axis). The bar corresponding to the best result per algorithm and problem is presented in dark gray shading, while the dotted line marks the accuracy of the baseline approach (without the CI component) for each problem.

One may easily observe that the CI component appears quite robust with regard to the γ parameter, as in all cases, it manages to outperform the baseline approach for at least 7 out the 10 possible parameter values. For the balanced *web activity* dataset, a value of $\gamma = 0.3$ yields the best results for both SS-LCS$_{\text{CI}}$ and UCS$_{\text{CI}}$, while for the imbalanced datasets UCS$_{\text{CI}}$ seems to benefit from more numerous initial populations (greater γ values).Overall, as a rule-of-thumb, a value of $\gamma = 0.3$ seems to be a good choice in all cases, yielding the best result for all problems with SS-LCS$_{\text{CI}}$ and acceptable accuracy rates (well above the baseline) with UCS$_{\text{CI}}$.

Figure 3 depicts the results of the second set of experiments, where the "uniform-distribution" strategy is in place. The balanced *web activity* dataset is not included, as the change in the distribution strategy does not affect the rule numerosities in its initial population.

Inspecting the obtained results, we observe that the CI component equipped with the "uniform-distribution" strategy (SS-LCS$_{CIu}$ and UCS$_{CIu}$, respectively, for the two studied LCS) appears extremely robust to the choice of γ values, achieving better results than the baseline approaches in all but two cases ($\gamma = 0.9$ for the *hepatitis* problem and $\gamma = 0.7$ for the *car* problem). The best achieved accuracy values in all cases are also higher than the corresponding results with the default rule distribution strategy (Fig. 2). Finally, a value of $\gamma = 0.5$ appears to be a



(a) SS-LCS$_{CIu}$ in the *hepatitis* dataset

(b) UCS$_{CIu}$ in the *hepatitis* dataset

(c) SS-LCS$_{CIu}$ in the *car* dataset

(d) UCS$_{CIu}$ in the *car* dataset

Fig. 3 Average classification accuracy (over 5 tenfold cross-validation runs) for SS-LCS$_{CIu}$ and UCS$_{CIu}$. For each problem-algorithm pair, results for various values of the γ parameter (x-axis) are reported. The *dotted line* marks the accuracy of the baseline approach (without the CI component) for each problem, as measured in the experiments of Table 5

good choice for all problems, yielding results well above the baseline and leading to the best solutions, in terms of predictive accuracy, in 3 out of the 4 studied problem-algorithm combinations.

To sum up, although the experiments presented in this Section serve only as an initial investigation towards better understanding the workings of the proposed CI method, there are some conclusions to be drawn. First, and more important, results indicate that the variability of results due to the γ parameter is small, thus relieving the user of the task of carefully tuning an additional parameter. Additionally, the "uniform-distribution" strategy shows promise of further boosting the studied LCS algorithms' performance in imbalanced domains, while still maintaining the small variability of results with respect to the γ parameter.

7 Conclusions and further work

The present paper investigates an initialization method for supervised LCS that leverages "summary information", extracted by clustering the target classification problem's training dataset, to produce candidate solutions for the algorithm's initial ruleset, prior to its exploration phase. Unlike most Michigan-style LCS, the studied algorithms, namely SS-LCS and UCS, are equipped with the proposed clustering-based initialization (CI) component that allows them to combine *pre-training* initialization with the traditional covering operator applied *during training*.

After describing the CI process and our design and implementation choices for integrating it to a supervised LCS framework, we provide the specifics of transforming identified clusters (by extracting information from their centroids and instance assignments) into rules. We then define and conduct an extended experimental investigation of its applicability and potential. Through two sets of experiments, designed to test the predictive accuracy and the expressive abilities of our newly proposed approach, we study the two versions (with and without the CI component) of our target algorithms and extend their comparison by (i) taking into account the time required to reach solutions and (ii) including five additional rule-based ML methods, namely C4.5, PART and the evolutionary algorithms GAssist, SIA and HIDER.

The obtained experimental results confirm our initial hypotheses that the CI component can boost LCS performance, both in terms of predictive accuracy and the final evolved ruleset's size. In the 20 studied classification problems

- SS-LCS$_{\text{CI}}$ outperforms its baseline approach at a level of confidence of 96.53%, when optimizing the achieved accuracy rate is the primary goal, while it also produces more compact rulesets at a level of confidence greater than 97%. On the other hand,
- UCS$_{\text{CI}}$ outperforms the corresponding non-CI version at a level of confidence greater than 90%, both in terms of predictive accuracy and evolved model size.

Although this increase in solution quality and readability would have sufficed to show an advantage, the CI versions of both studied algorithms also do better than the baseline versions in terms of convergence speed, with an average 26% improvement of training times required to reach the best solution, when optimizing predictive accuracy and a 10% improvement for ruleset size. Moreover, they compare well against all rivals in both sets of conducted experiments, achieving high average ranks according to the Friedman test. Additional experiments aimed at better understanding the proposed method's behavior, in relation to specific parameter values and design choices, reveal its robustness and point towards further possible performance

improvements in domains where high class imbalances or within-class schema imbalances exist.

Overall, we consider this initial investigation successful, as it provides clear indications that LCS can profit from a carefully designed pre-training initialization procedure and paves the way for further investigations in this direction. The main direction along which we intend to continue our work is the systematic study of the CI method's parameters, including the number of clusters γ and the generalization probabilities $P_{\#}^{ci}$, to fully assess their effect on system performance. We also intend to investigate alternative design choices for the CI method, ranging from the clustering algorithm used to the way clusters are transformed into rules, with the ultimate goal of providing a robust general-use initialization component for the class of supervised LCS algorithms.

Acknowledgments This paper is part of the 03ED735 research project, implemented within the framework of the "Reinforcement Programme of Human Research Manpower" (PENED) and cofinanced by National and Community Funds (25% from the Greek Ministry of Development-General Secretariat of Research and Technology and 75% from E.U.-European Social Funding).

References

Aguilar-Ruiz JS, Riquelme JC, Toro M (2003) Evolutionary learning of hierarchical decision rules. IEEE Trans Syst Man Cybern B 33(2):324–331

Aguilar-Ruiz J, Giraldez R, Riquelme J (2007) Natural encoding for evolutionary supervised learning. IEEE Trans Evol Comput 11(4):466–479. doi:10.1109/TEVC.2006.883466

Alcalá-Fdez J, Sánchez L, García S, del Jesus M, Ventura S, Garrell J, Otero J, Romero C, Bacardit J, Rivas V, Fernández J, Herrera F (2009) Keel: a software tool to assess evolutionary algorithms for data mining problems. Soft Comput 13:307–318. doi:10.1007/s00500-008-0323-y

Asuncion A, Newman DJ (2010) UCI machine learning repository. University of California, School of Information and Computer Science, Irvine. http://archive.ics.uci.edu/ml

Bacardit J (2004) Pittsburgh genetics-based machine learning in the data mining era: representations, generalization, and run-time. PhD thesis, Ramon Llull University, Barcelona, Catalonia, Spain

Bacardit J (2005) Analysis of the initialization stage of a Pittsburgh approach learning classifier system. In: Proceedings of the 2005 conference on genetic and evolutionary computation (GECCO '05). ACM, New York, pp 1843–1850

Bernadó-Mansilla E, Garrell-Guiu JM (2003) Accuracy-based learning classifier systems: models, analysis and applications to classification tasks. Evol Comput 11(3):209–238. doi:10.1162/106365603322365289

Bonelli P, Parodi A, Sen S, Wilson SW (1990) NEWBOOLE: a fast GBML system. In: Proceedings of the 7th international conference on machine learning. Morgan Kaufmann, San Francisco, pp 153–159

Breiman L (2002) Wald Lecture II—looking inside the black box. In: 277th meeting of the Institute of Mathematical Statistics

Burke EK, Newall JP, Weare RF (1998) Initialization strategies and diversity in evolutionary timetabling. Evol Comput 6:81–103

Butz MV, Wilson SW (2001) An algorithmic description of XCS. In: IWLCS '00: revised papers from the third international workshop on advances in learning classifier systems. Springer, London, pp 253–272

Butz MV, Kovacs T, Lanzi PL, Wilson SW (2004) Toward a theory of generalization and learning in XCS. IEEE Trans Evol Comput 8(1):28–46

Butz MV, Sastry K, Goldberg DE (2005) Strong, stable, and reliable fitness pressure in XCS due to tournament selection. Genet Program Evol Mach 6(1):53–77. doi:10.1007/s10710-005-7619-9

Chou CH, Chen JN (2000) Genetic algorithms: initialization schemes and genes extraction. In: The ninth IEEE international conference on fuzzy systems, 2000 (FUZZ IEEE 2000), vol 2, pp 965–968

De Jong KA (1975) An analysis of the behavior of a class of genetic adaptive systems. PhD thesis, University of Michigan, Ann Arbor, MI, USA

De Jong KA, Spears WM, Gordon DF (1993) Using genetic algorithms for concept learning. Mach Learn 13:161–188. doi:10.1007/BF00993042

Demšar J (2006) Statistical comparisons of classifiers over multiple data sets. J Mach Learn Res 7:1–30

Dixon PW, Corne DW, Oates MJ (2003) A ruleset reduction algorithm for the XCS learning classifier system. In: Learning classifier systems, 5th international workshop, IWLCS 2002, Granada, Spain, September 7–8, 2002, revised papers. Lecture notes in computer science, vol 2661. Springer, Berlin-Heidelberg, pp 20–29

Frank E, Witten IH (1998) Generating accurate rule sets without global optimization. In: ICML '98: proceedings of the fifteenth international conference on machine learning. Morgan Kaufmann, San Francisco, pp 144–151

Friedman M (1937) The use of ranks to avoid the assumption of normality implicit in the analysis of variance. J Am Stat Assoc 32(200):675–701

Friedman M (1940) A comparison of alternative tests of significance for the problem of m rankings. Ann Math Stat 11(1):86–92

García S, Fernández A, Luengo J, Herrera F (2009) A study of statistical techniques and performance measures for genetics-based machine learning: accuracy and interpretability. Soft Comput 13:959–977. doi:10.1007/s00500-008-0392-y

González A, Pére R (1999) Slave: a genetic learning system based on an iterative approach. IEEE Trans Fuzzy Syst 7(2):176–191. doi:10.1109/91.755399

Guan SU, Zhu F (2005) An incremental approach to genetic-algorithms-based classification. IEEE Trans Syst Man Cybern B: Cybern 35(2):227–239. doi:10.1109/TSMCB.2004.842247

Holland JH (1975) Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control and artificial intelligence. University of Michigan Press, Ann Arbor

Holmes JH, Sager JA (2005) Rule discovery in epidemiologic surveillance data using EpiXCS: an evolutionary computation approach. In: Miksch S, Hunter J, Keravnou ET (eds) AIME. Lecture notes in computer science, vol 3581. Springer, Heidelberg, pp 444–452

Ishibuchi H, Yamamoto T, Nakashima T (2005) Hybridization of fuzzy gbml approaches for pattern classification problems. IEEE Trans Syst Man Cybern B: Cybern 35(2):359–365. doi:10.1109/TSMCB.2004.842257

Janikow CZ (1992) Inductive learning of decision rules from attribute-based examples: a knowledge-intensive genetic algorithm approach. PhD thesis, University of North Carolina at Chapel Hill, Chapel Hill, NC, USA

Kallel L, Schoenauer M (1997) Alternative random initialization in genetic algorithms. In: Proceedings of the 7th international

conference on genetic algorithms. Morgan Kaufmann, pp 268–275
Kang RG, Jung CY (2006) The improved initialization method of genetic algorithm for solving the optimization problem. In: King I, Wang J, Chan LW, Wang D (eds) Neural information processing. Lecture notes in computer science, vol 4234. Springer, Berlin-Heidelberg, pp 789–796
Kovacs T (1999) Deletion schemes for classifier systems. In: Banzhaf W, Daida J, Eiben AE, Garzon MH, Honavar V, Jakiela M, Smith RE (eds) Proceedings of the genetic and evolutionary computation conference (GECCO-99). Morgan Kaufmann, San Francisco, pp. 329–336
Kovacs T (2002a) XCS's strength-based twin: part I. In: Lanzi et al (2003), pp 61–80
Kovacs T (2002b) XCS's strength-based twin: part II. In: Lanzi et al (2003), pp 81–98
Lanzi PL (2008) Learning classifier systems: then and now. Evol Intell 1(1):63–82
Lanzi PL, Stolzmann W, Wilson SW (eds) (2003) Learning classifier systems. 5th international workshop, IWLCS 2002, Granada, Spain, September 7–8, 2002, revised papers. Lecture notes in computer science, vol 2661. Springer, Berlin
Louis SJ, McDonnell J (2004) Learning with case-injected genetic algorithms. IEEE Trans Evol Comput 8(4):316–328
Maaranen H, Miettinen K, Mäkelä MM (2004) Quasi-random initial population for genetic algorithms. Comput Math Appl 47(12):1885–1895
Mitchell TM (1997) Machine learning. McGraw-Hill Higher Education
Nemenyi PB (1963) Distribution-free multiple comparisons. PhD thesis, Princeton University
Orriols-Puig A, Bernadó-Mansilla E (2008a) Mining imbalanced data with learning classifier systems. In: Bull L, Bernadó-Mansilla E, Holmes JH (eds) Learning classifier systems in data mining. Studies in computational intelligence, vol 125. Springer, Berlin, pp 123–145. doi:10.1007/978-3-540-78979-6_6
Orriols-Puig A, Bernadó-Mansilla E (2008b) Revisiting UCS: description, fitness sharing, and comparison with XCS. Learning classifier systems: 10th international workshop, IWLCS 2006, Seattle, MA, USA, July 8, 2006 and 11th international workshop, IWLCS 2007, London, UK, July 8, 2007, revised selected papers, pp 96–116. doi:10.1007/978-3-540-88138-4_6
Orriols-Puig A, Goldberg DE, Sastry K, Bernadó-Mansilla E (2007) Modeling XCS in class imbalances: population size and parameter settings. In: GECCO '07: proceedings of the 9th annual conference on Genetic and evolutionary computation. ACM, New York, pp 1838–1845. doi:10.1145/1276958.1277324
Orriols-Puig A, Casillas J, Bernadó-Mansilla E (2008) Genetic-based machine learning systems are competitive for pattern recognition. Evol Intell 1:209–232. doi:10.1007/s12065-008-0013-9
Orriols-Puig A, Casillas J, Bernadó-Mansilla E (2009) Fuzzy-UCS: a Michigan-style learning fuzzy-classifier system for supervised learning. IEEE Trans Evol Comput 13(2):260–283
Quinlan JR (1993) C4.5: programs for machine learning. Morgan Kaufmann, San Francisco
Quinlan JR (1996) Learning first-order definitions of functions. J Artif Intell Res 5:139–161
Rahnamayan S, Tizhoosh HR, Salama MMA (2007) A novel population initialization method for accelerating evolutionary algorithms. Comput Math Appl 53(10):1605–1614
Ramsey CL, Grefenstette JJ (1993) Case-based initialization of genetic algorithms. In: Proceedings of the 5th international conference on genetic algorithms. Morgan Kaufmann, San Francisco, pp 84–91
Tzima F, Mitkas P (2010) Comparing strength and accuracy-based supervised learning classifier systems. Technical report, Intelligent Systems and Software Engineering Labgroup, Department of Electrical and Computer Engineering, Aristotle University of Thessaloniki, Thessaloniki, Greece, GR-541 24
Tzima FA, Mitkas PA, Voukantsis D, Karatzas KD (2011) Sparse episode identification in environmental datasets: the case of air quality assessment. Expert Syst Appl 38(5):5019–5027
Vavliakis KN, Symeonidis AL, Mitkas PA (2010) Towards understanding how personality, motivation, and events trigger Web user activity. In: IEEE/WIC/ACM international conference on Web intelligence and intelligent agent technology
Venturini G (1993) Sia: a supervised inductive algorithm with genetic search for learning attributes based concepts. In: Proceedings of the European conference on machine learning. Springer, London, pp 280–296
Wilcoxon F (1945) Individual comparisons by ranking methods. Biom Bull 1(6):80–83
Wilson SW (1994) ZCS: A zeroth-level classifier system. Evol Comput 2(1):1–18
Wilson SW (1995) Classifier fitness based on accuracy. Evol Comput 3(2):149–175
Wilson SW (2002) Compact rulesets from XCSI. In: IWLCS '01: revised papers from the 4th international workshop on advances in learning classifier systems. Springer, London, pp 197–210
Witten IH, Frank E (2005) Data mining: practical machine learning tools and techniques, 2nd edn. Morgan Kaufmann, San Francisco
Zhang G, Gao L, Shi Y (2011) An effective genetic algorithm for the flexible job-shop scheduling problem. Expert Syst Appl 38(4): 3563–3573