Memetic Computing manuscript No.
(will be inserted by the editor)

Integrating memetic search into the BioHEL evolutionary learning system for large-scale datasets

Dan Andrei Calian · **Jaume Bacardit**

Received: date / Accepted: date

Abstract Local search methods are widely used to improve the performance of evolutionary computation algorithms in all kinds of domains. Employing advanced and efficient exploration mechanisms becomes crucial in complex and very large (in terms of search space) problems, such as when employing evolutionary algorithms to large-scale data mining tasks. Recently, the GAssist Pittsburgh evolutionary learning system was extended with memetic operators for discrete representations that use information from the supervised learning process to heuristically edit classification rules and rule sets. In this paper we first adapt some of these operators to BioHEL, a different evolutionary learning system applying the iterative learning approach, and afterwards propose versions of these operators designed for continuous attributes and for dealing with noise. The performance of all these operators and their combination is extensively evaluated on a broad range of synthetic large-scale datasets to identify the settings that present the best balance between efficiency and accuracy. Finally, the identified best configurations are compared with other classes of machine learning methods on both synthetic and real-world large-scale datasets and show very competent performance.

Keywords Memetic Algorithms · Evolutionary Algorithms · Evolutionary Rule Learning

D. A. Calian
Department of Computer Science, University College London, Gower Street, WC1E 6BT, London, UK
E-mail: d.calian@cs.ucl.ac.uk

J. Bacardit
Interdisciplinary Computing and Complex Systems (ICOS) research group, School of Computer Science, University of Nottingham, Jubilee Campus, Wollaton Road, NG8 1BB, Nottingham, UK
E-mail: jaume.bacardit@nottingham.ac.uk

1 Introduction

The creation of advanced/smart exploration mechanisms has been a very active topic of research for many years in evolutionary computation and other related fields such as swarm intelligence, with the proposal of several paradigms such as Memetic algorithms (MA) [27], Estimation of distribution algorithms (EDA) [28], Ant Colony Optimisation [12], Particle Swarm Optimisation [25] or Genetic Programming [26], to name just a few.

In the specific context of evolutionary rule learning, most of these advances are concentrated in two of these families of methods: MA and EDA. EDA techniques estimate a model for the structure of the problem and guide the evolutionary system towards creating offspring according to this model. Memetic algorithms are based on integrating local search mechanisms within global search. Recently, both of these kinds of advanced exploration have been adopted within the specific context of evolutionary learning, with examples of EDA-based exploration mechanisms [9,31] as well as Memetic methods [6,35].

Previously [6] we have shown that integrating memetic operators within the GA cycle of the GAssist Pittsburgh style LCS [1] has positive outcomes, enabling the system to learn more compact rule sets, in less time, while being more robust to noise. GAssist was hybridized with local search using two types of operators: rule-set-wise operators, which given several sets of rules as input heuristically select the minimum subset of rules that obtain maximum accuracy and rule-wise operators for discrete attributes, which edit individual rules to improve their accuracy.

BioHEL [2] is an evolutionary learning system which follows the iterative rule learning approach, whose design and implementation have been strongly influenced by GAssist. Because both GAssist and BioHEL work with the same type of individual rules, the rule-wise operators of GAssist

are directly portable to BioHEL. But given that BioHEL maintains only one rule set (representing the current active solution) and GAssist's rule-set-wise operator requires at least two rule sets as input, this latter operator can not be directly applied to BioHEL.

This paper continues on this line of research by incorporating GAssist's memetic operators for discrete attributes into BioHEL and by defining and evaluating new rule-wise operators for continuous attributes. These continuous-attributes operators have been tailored to the attribute-list knowledge representation (ALKR) [2,5] of BioHEL and integrated into the system. To improve BioHEL's performance on noisy datasets, the continuous-attributes operators have been also adapted to cope with such kind of data.

The new and adapted operators (and their combination) are extensively evaluated on a broad range of synthetic and real-world large-scale classification problems using two different criteria: efficiency and accuracy, while also identifying the combination of operators with the best balance between both criteria. Finally, the performance of the best settings of the Memetic BioHEL is compared to the performance of other machine learning methods.

The rest of the paper is structured as follows: section 2 overviews related work, section 3 describes the BioHEL system in more detail and the existing memetic operators for discrete representations. Section 4 defines and describes in detail the six new continuous-attributes rule wise local search operators integrated within BioHEL. The experimental design used is described in section 5 while section 6 contains the results obtained and a discussion of these. Finally, section 7 contains the conclusions and directions for future work.

2 Related work

Local search, statistical learning techniques and intelligent recombination are some of the methods used to improve the efficiency and performance of evolutionary computation algorithms. The literature provides different types of techniques which are based on these strategies.

Memetic algorithms [27], which also include the research presented in this paper, encompass all evolutionary algorithms which hybridise global with local search, or in which individual refinement takes place. These algorithms are inspired by models of natural systems, where individuals not only evolve from a genetic perspective but also evolve by adopting cultural units of transmission.

An early example of memetic algorithms in Learning Classifier Systems represents the SAMUEL system [20], which has used an operator similar to the rule-set-wise (RSW) operator from our earlier work in MPLCS [6]. SAMUEL has been applied to multi-step domains and its operator produced offspring composed from high-payoff rules which fired in sequence. A difference between RSW and this LS operator is that the former has been designed to use rules from N parents and creates an ordered rule set, while the latter can only use rules from 2 parents and generates an unordered rule set. A more recent work in the LCS literature proposed a gradient descent method for improving the performance of XCS [34] on multi-step domains. Another work proposed a Memetic Learning Classifier System within the Michigan approach of LCS [35] concerning real-valued representations; in this study, XCS has been hybridised with Lamarckian Evolution-based memetic operators.

Estimation of distribution algorithms (EDA) are another family of techniques used for improving evolutionary computation algorithms. These employ machine learning or statistical techniques to build a model of the structure of the problem being solved in order to guide the evolutionary search towards better solutions through the use of informed exploration operators. In [9] XCS has been extended with a crossover operator based on two types of estimation of distribution algorithms: the Extended Compact Genetic Algorithm [22] and the Bayesian Optimisation Algorithm [32]. Both methods use the best rules of the population to derive global structural information which is later used to inform the crossover operator.

A recent integration of EDAs within a Pittsburgh LCS represents the Compact Classifier System (CCS) [30]. CCS uses the Compact Genetic Algorithm (CGA) [23] which evolves one rule at a time. So to generate complete solutions, the CGA is iteratively run to produce different rules. Each CGA run starts from a different perturbation of the same initial solution, where each CCS individual stores a set of such perturbations. The goal of the CCS is to evolve the minimal set of rules which compose a maximally general solution. More recently, $\chi eCCS$, an extension of CCS was proposed in [31]. The $\chi eCCS$ generates a population of rules using the recombination mechanics and model building techniques of the extended compact genetic algorithm [22].

3 Background material

In this section we provide a general description of the BioHEL evolutionary system and detail its characteristics that are relevant to this paper. We also describe GAssist's local search operators for discrete attributes as these form the basis for the new operators for continuous attributes presented in this paper.

3.1 BioHEL

BioHEL [2] is an evolutionary learning system belonging to the iterative rule learning approach (IRL), first used in the field of evolutionary learning in [33]. As mentioned in the

introduction, this system is a descendant of the GAssist LCS [1] keeping most of its core modules.

BioHEL generates the solution to a problem as a rule set structured as a decision list. This list is constructed by iteratively applying a genetic algorithm (GA) to produce individual rules which are added to the rule set. When a rule is added all the examples covered by it are removed from the training set, to focus the search on different parts of the search space. The system uses an explicit default rule located at the end of the rule set having a predefined class as default. Hence, the iterative rule learning process finishes when the GA cannot generate a rule that is better than just assigning any remaining training examples to the default class. BioHEL's fitness function is based on the Minimum Description Length principle, and it is designed to reward rules with high accuracy, high coverage and low complexity. For a complete description of the fitness function please see [2].

Given that in this paper we are proposing memetic operators to edit rules, it is necessary to describe in detail BioHEL's knowledge representation: the Attribute List Knowledge Representation (ALKR) [2]. ALKR is a meta-representation that is designed to automatically identify and represent only the relevant information of the problem. To this aim, for each individual in the population this representation will contain (a) a list of relevant attributes and (b) for each of these attributes, its associated predicate in the rule and (c) its associated class. Two operators are added to the GA cycle to probabilistically add (specialize) or remove (generalize) attributes from an individual's list. The list of relevant attributes can be different for each individual in the population, hence this mechanism is more than just a global feature selection process.

Depending on the nature of each of the attributes in the list (discrete or continuous), the associated predicate will differ. For discrete attributes the GABIL [10] representation is employed. For continuous ones we employ a simple intervalar representation [29] (where an interval over the domains of each of the relevant attributes is defined, with a lower and an upper bound).

3.2 Memetic operators for discrete attributes

An in-depth presentation of the design and evaluation of three rule-wise memetic operators for discrete attributes incorporated within the GAssist Pittsburgh system is given in [6]. These operators have been specifically tailored to the GABIL knowledge representation [10]. Since these operators have formed the basis behind the reasoning for the new continuous operators, it is important that they are described. The three rule-wise memetic operators for discrete attributes are:

Rule Cleaning (RC)
The rule cleaning operator edits a rule making it cover less negative examples (which are the ones misclassified by the rule). To do so it identifies the terms in the GABIL representation that only cover negative examples, but no positive ones, and disables the term which will make the rule drop the largest number of negative examples.

Rule Splitting (RS)
In many cases the RC operator cannot be successfully applied because GABIL terms cover both positive and negative examples. To alleviate this issue RS splits a rule into two rules (by logically decomposing the rule's predicate into a disjunction of two predicates) in such a way that the rule cleaning operator can be successfully applied to one of the sub-rules. The split point and the subsequent cleaned term are chosen, as for RC, to maximise the number of dropped negative examples.

Rule Generalizing (RG)
The rule generalizing operator edits a rule to make it cover more positive examples than what it initially covered, without classifying any new negative examples. To do so, all disabled literals in a rule are tested to determine which is the one that will cover the greatest number of new positive examples without any misclassifications.

The order in which these operators are applied to a rule is important. The RS operator has been designed to be applied to rules which the RC operator cannot fix, and so rule splitting is applied after rule cleaning. Furthermore, the RG operator will always be applied after the other two operators. The idea behind this decision is that the specificity pressure (by forcing the rule to cover fewer examples) introduced by RC and RS must be compensated with the generality pressure (by making the rule cover more examples) introduced by RG. For a full description of these operators, including examples, pseudo-code and evaluation, please see [6].

4 Memetic extensions of BioHEL

This section is concerned with the design and integration of memetic operators within the BioHEL evolutionary learning system. The next subsections overview the integration procedure, the design of the new three memetic operators for non-noisy continuous-attributes and the design of the three noise-tolerant continuous-attributes local search operators.

4.1 Integrating GAssist's memetic operators within BioHEL

As mentioned in the introduction, GAssist had two classes of Memetic operators, the rule-wise ones (described in the previous section) and a rule set-wise operator, which integrated knowledge from multiple rule sets to create a new,

single, rule set. Given that individuals in BioHEL are rules, only the former class of operators will be adapted to the new system. The operators will be applied to each of the individuals in BioHEL's population with a certain individual-wise probability after the mutation stage of the GA cycle but before the elitism stage.

For problems with discrete attributes only, the exact same RC and RG operators used in GAssist will be employed here, only adapting them to the usage of the ALKR representation. For RS, however, we have to do an adaptation. The application of this operator produces two rules. In GAssist the two rules were inserted into an individual (a rule set). Given that in BioHEL each individual is a rule we cannot follow the same procedure. Instead we will do the following: In the local search stage of the GA cycle the selected memetic operators will be applied iteratively to the population given a probability, as described above. For each individual that has RS applied to it, one of the resulting rules will replace the individual, while the other rule will be kept in a separate list. After the memetic operators have been applied to the entire population, the rules from this separate list will replace the worst individuals of the population.

4.2 Memetic operators for continuous attributes

As in the case for the discrete-data operators, the continuous-attributes operators RC and RS edit the rules to make them more specific while RG edits the rules to make them more general. All three memetic operators function by requiring as input (besides the rule to edit) information about the matched and unmatched instances of the rule. This information is generated by two other algorithms. The first algorithm, *the M&U operator*, computes arrays of matched and unmatched instances and sorts them by the domain value of each attribute in the rule, while the second one, *the CDS operator*, compiles together these sorted instances based on their class value and assigns them a label identifying the mix of classes within the group. Fig. 1 shows how the instances are preprocessed by the M&U and CDS operators, before being passed as input to the continuous attributes memetic operators. Before detailing how the three continuous-attributes memetic operators improve rules, the mode of operation of these two preprocessing algorithms is described in detail.

4.2.1 Computing matched and unmatched instances (M&U operator)

The M&U operator when applied to a rule first computes the matched or unmatched instances of the rule. Afterwards, for each expressed attribute of the rule, it builds a list of these computed instances sorted by the value of the current attribute. RC and RS require sorted and clustered arrays of the matched examples, while RG requires sorted and clustered arrays of unmatched examples. For RG, the sorted unmatched examples are computed separately for each attribute in such a way to only contain the examples that failed to be matched only by the attribute in question. Also, for each attribute, these unmatched examples are split into two disjoint sets: the instances with the domain value less than the lower bound of the rule's attribute interval (also referred to as *left* unmatched instances), and the ones with a domain value greater than the upper bound of the rule's attribute interval (also referred to as *right* unmatched instances).

The way in which the matched and unmatched examples are computed is better illustrated using an example:

- Given the following instances (the domain values of two attributes separated by a comma, followed by $\|$ and the class):
  a: -1.0, 1.2 $\|$ 1
  b: 1.2, 2.8 $\|$ 0
  c: 0.1, 0.0 $\|$ 0
  d: 0.1, 1.6 $\|$ 1
  e: -4.0, 5.3 $\|$ 1
  f: 1.7, 0.3 $\|$ 0
  g: -2.3, -3.6 $\|$ 0
  h: 1.3, -3.2 $\|$ 0
  i: 3.4, 1.8 $\|$ 1
  j: -3.5, -1.5 $\|$ 0
  k: -3.0, 0.9 $\|$ 1
  l: -1.0, -5.6 $\|$ 0
- And the following rule: $[-2.0, 2.0], [-2.0, 2.0] \mapsto 0$
- The matched instances are computed and sorted:
  on the domain value of the first attribute:
  a: -1.0, 1.2 $\|$ 1
  c: 0.1, 0.0 $\|$ 0
  d: 0.1, 1.6 $\|$ 1
  f: 1.7, 0.3 $\|$ 0
  and on the domain value of the second attribute:
  c: 0.1, 0.0 $\|$ 0
  f: 1.7, 0.3 $\|$ 0
  a: -1.0, 1.2 $\|$ 1
  d: 0.1, 1.6 $\|$ 1
- Then the unmatched instances which only failed to be matched by the attribute in question are computed and sorted:
  for the first attribute:
  domain value less than the lower bound of the rule's attribute interval ($<$ -2.0) or simply *left* unmatched instances:
  j: -3.5, -1.5 $\|$ 0
  k: -3.0, 0.9 $\|$ 1
  domain value greater than the upper bound of the rule's attribute interval ($>$ 2.0) or simply *right* unmatched instances:
  i: 3.4, 1.8 $\|$ 1
  and for the second attribute:

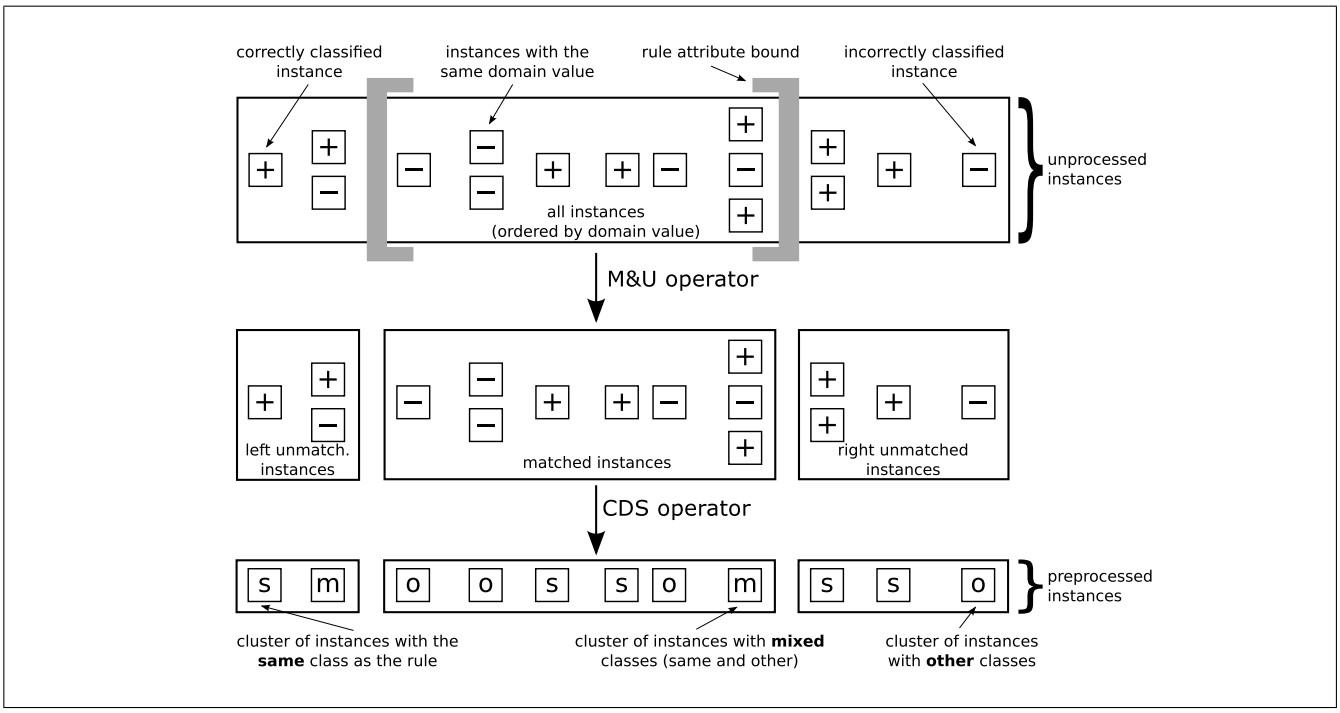


Fig. 1: The pre-processing steps

left unmatched instances ($<$ -2.0):
l: -1.0, -5.6 || 0
h: 1.3, -3.2 || 0
right unmatched instances ($>$ 2.0):
b: 1.2, 2.8 || 0

4.2.2 Clustering of domain segments (the CDS operator)

After the arrays of matched and unmatched examples have been computed by the M&U operator, the CDS operator is applied to cluster the matches and unmatches. This second pre-processing step is needed to ensure that in case more instances have the same domain value but belong to different classes, the memetic operators do not mistakenly add negative examples to or remove positive ones from a rule's cover set (which is the set composed of the rule's matched instances). Clustering an array of instances refers to grouping together all instances with the same domain value for a given attribute along with assigning them a label which identifies the mix of classes within the group. The number of instances and the domain value of each group are also stored to be able to compare different cleaning or generalizing possibilities. The labels assigned to the groups of instances are: *same* (when all the instances within the group have the same class as the rule), *other* (when all the instances within the group have different classes from the rule) and *mixed* (when at least one of the instances within the group has the same class as the rule).

The following example illustrates this process:

- Given any rule which predicts class 0.
- And the following list of instances sorted on the first attribute:
  a: -1.5, 1.2 || 0
  b: -1.2, 2.8 || 1
  c: -1.2, 0.0 || 1
  d: 2.1, 1.6 || 1
  e: 4.4, 5.3 || 0
  f: 4.4, 0.3 || 1
  g: 4.4, -3.6 || 0
  h: 5.3, -3.2 || 0
- The result of applying the CDS operator on these instances is:
  a: *same*
  b,c: *other*
  d: *other*
  e,f,g: *mixed*
  h: *same*

All three memetic operators edit rules by modifying the lower, the upper or both bounds of one attribute interval of each rule. Fig. 2 graphically depicts the mode of operation of the three continuous-data memetic operators.

4.2.3 Rule Cleaning

The rule cleaning operator is applied to a rule based on the sorted matched examples covered by this rule, and heuristically shrinks the domain interval of the expressed attribute which removes the largest number of misclassified examples from the rule's cover set without changing its correctly clas-

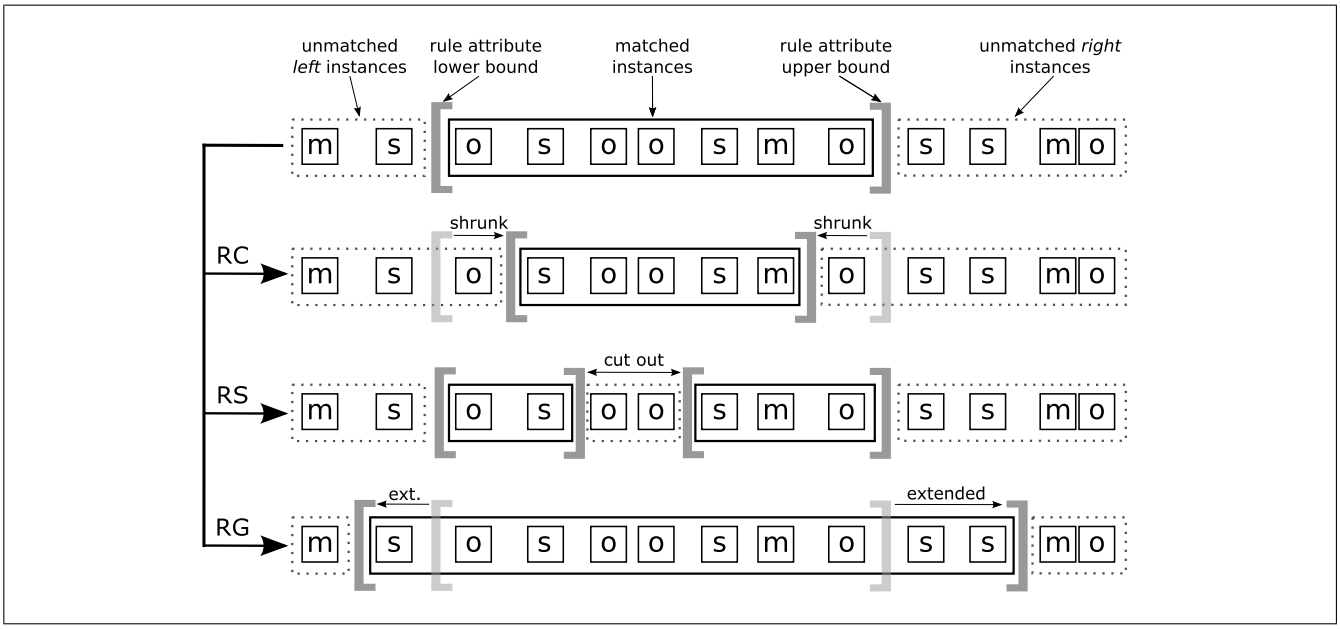


Fig. 2: The mode of operation of the three memetic operators

sified instances. The following example illustrates how the RC local search operates:

- Given the following rule: $[-2.0, 2.0], [-2.0, 2.0] \mapsto 0$
- And the matched instances of the rule sorted on the first attribute (not also clustered using the CDS operator):
  a: -1.0, 1.5 || 1
  b: 0.1, 0.6 || 0
  c: 0.1, 0.7 || 1
  d: 1.2, 1.6 || 0
  e: 1.7, 0.4 || 0
- The rule classifies correctly three examples and incorrectly two (a and c).
- Clustering the sorted matched instances we get:
  a: *other*
  b,c: *mixed*
  d: *same*
  e: *same*
- By shifting the lower bound of the first attribute's interval of the rule from -2 to 0.1 the rule will correctly classify three examples and incorrectly just one (b). The 0.1 is the domain value of the first group (in this case b,c) with a *mixed* or a *same* label found after the longest streak of groups with an *other* label while traversing the clustered groups from top to bottom.

Thus the rule cleaning procedure finds, for each expressed attribute, the smallest interval which enables the rule to cover the same positively matched examples while covering the lowest number of misclassified examples. It does this by iterating through the clustered sorted matches from top to bottom (for the attribute interval's lower bound and in reverse for the upper bound) and stopping when the label of the current group is either *same* or *mixed*; meaning that all of the previously checked groups had an *other* label and thus contained only misclassified examples. Then, new lower and/or upper bounds are built using the domain values of the last visited groups. All the different possibilities for cleaning a rule are enumerated, but only the pair of bounds which would remove the highest number of misclassifications from the cover set of the rule are actually substituted with their respective counterparts; meaning that only one clean operation is performed. Fig. 3 contains the pseudo-code for this operator.

4.2.4 Rule Splitting

The rule splitting operator is applied to a rule based on the sorted matched examples covered by this rule, and heuristically splits a rule into two on the attribute interval which allows the two subrules, when taken together, to misclassify a lower number of instances as compared to the initial rule. The motivation and rationale of this operator are better illustrated through an example:

- Given the following rule: $[-2.0, 2.0], [-2.0, 2.0] \mapsto 0$
- And the matched instances of the rule sorted on the first attribute (not also clustered using the CDS operator):
  a: -0.7, 0.2 || 1
  b: 0.5, 1.8 || 0
  c: 1.1, 1.2 || 1
  d: 1.1, 0.0 || 1
  e: 1.8, 0.3 || 1
  f: 2.0, 1.5 || 0
- The rule classifies correctly two examples and incorrectly four (a, c, d and e).

```
Procedure Rule Cleaning operator
Input: rule, matchedExamplesPerAttribute

maxNumberOfNegativeMatches = 0
Foreach att in numAttributes
    bothLeftAndRight = 0
    tempPosLeft = tempPosRight = -1

    pos = 0
    While pos < getNumClusteredMatchesOfAtt(att) and
        getClusteredMatchOfAtt(att, pos).label == other
        pos ++
        bothLeftAndRight += getClusteredMatchOfAtt(att, pos).length
    EndWhile
    If pos > 0
        tempPosLeft = pos
    EndIf

    pos = getNumClusteredMatchesOfAtt(att) - 1
    While pos > -1 and getClusteredMatchOfAtt(att, pos).label == other
        pos --
        bothLeftAndRight += getClusteredMatchOfAtt(att, pos).length
    EndWhile
    If getNumClusteredMatchesOfAttr(att) - 1 - pos > 0
        tempPosRight = pos
    EndIf

    If bothLeftAndRight > maxNumberOfNegativeMatches and
      clean boundaries are not in reverse
        maxNumberOfNegativeMatches = bothLeftAndRight
        Remember tempPosLeft, tempPosRight and att in
                leftCPos, rightCPos and toClean
    EndIf
EndForeach

If maxNumberOfNegativeMatches > 0
    If can clean left
      rule.lowerBound = getClusteredMatchOfAtt(toClean, leftCPos).domVal
    EndIf
    If can clean right
      rule.upperBound = getClusteredMatchOfAtt(toClean, rightCPos).domVal
    EndIf
EndIf

Output: rule
```

Fig. 3: Pseudo-code of the RC operator

- Clustering the sorted matched instances we get:
  a: *other*
  b: *same*
  c,d: *other*
  e: *other*
  f: *same*
- To eliminate the adjacent misclassified examples we could construct two subrules from the original rule by splitting it into two on the interval of the first attribute and simply omitting the (0.5, 2.0) section of the interval which covers the misclassified examples c, d and e. Then, one of the rules would cover the examples a and b, having the interval of the first attribute set to [-2.0, 0.5], while the other would cover example f with the interval of the first attribute set to [2.0, 2.0]. Now, when taken together, these two subrules correctly classify the same number of instances (2) but missclassify only one example, instead of four.

When RS is applied to a rule it selects one attribute and generates two identical rules, except for the domain intervals of a selected attribute. For this selected attribute one subrule has the same lower bound as the initial rule but a lower upper bound while the other subrule has the same upper bound but a higher lower bound. The rule is split into two so that the section not covered by the union of the subrules' intervals would match only negative examples. This means that in order to apply the RS operator to a certain attribute, the sorted and clustered matched instances on that attribute must contain at least one group with an *other* label surrounded by at least two groups with different labels than *other*.

RS works by identifying, for each expressed attribute of the rule, the longest streak of adjacent groups with an *other* label from the list of sorted and clustered matched instances on the current attribute. Afterwards, the domain value of the first group located before the whole streak in the list becomes the first subrule's new upper bound and the domain value of the first group located after the whole streak becomes the second subrule's new lower bound. All possible splits are enumerated but only the split application which removes the greatest number of misclassified examples from the rule's cover set is actually applied. Fig. 4 contains the pseudo-code for this operator.

```
Procedure Rule Splitting operator
Input: rule, matchedExamplesPerAttribute

maxNumberOfNegativeMatches = 0
Foreach att in numAttributes
    tempMaxNumOfNegMs = 0
    tempStart = tempEnd = -1

    For pos = 0, pos < getNumClusteredMatchesOfAttr(att), pos ++
        If getClusteredMatchOfAtt(att,pos).label == other
            tempMaxNumOfNegMs+=getClusteredMatchOfAtt(att,pos).length
            tempEnd = pos
            If tempMaxNumOfNegMs == 0
                tempStart = pos
            EndIf
        Else (label is either mixed or same)
            If tempMaxNumOfNegMs > maxNumberOfNegativeMatches and
              tempStart > 0 and
              tempEnd <= getNumClusteredMatchesOfAttr(att) - 2
                maxNumberOfNegativeMatches = tempMaxNumOfNegMs
                Remember tempStart, tempEnd and att in
                        startCleanPos, endCleanPos and toSplit
            EndIf
            tempStart = tempEnd = -1
            tempMaxNumOfNegMs = 0
        EndIf
    EndFor
EndForeach

If maxNumberOfNegativeMatches > 0
    Create two subrules as copies of rule: r1, r2
    r1.upperBound = getClusteredMatchOfAtt(toSplit, startCleanPos - 1).domVal
    r2.lowerBound = getClusteredMatchOfAtt(toSplit, endCleanPos + 1).domVal
    Output: r1, r2
Else
    Output: rule
EndIf
```

Fig. 4: Pseudo-code of the RS operator

4.2.5 Rule Generalizing

The rule generalizing operator is applied to a rule based on the sorted unmatched examples covered by the rule, and heuristically enlarges one of the rule's expressed attribute interval which makes the rule cover the maximal number of new positive examples without covering any new negative examples.

How RG operates is best shown using an example:

- Given the following rule: $[-2.0, 2.0], [-2.0, 2.0] \mapsto 0$
- The unmatched instances which only failed to be matched by the first attribute are computed and sorted, for example:
  left ones ($<$ -2.0):
    a: -3.5, 0.9 $\|$ 1
    b: -3.5, -1.5 $\|$ 0
    c: -2.5, 1.2 $\|$ 0
  right ones ($>$ 2.0):
    d: 3.4, 1.8 $\|$ 1
    e: 3.4, 1.8 $\|$ 1
- Then clustered:
  left ones ($<$ -2.0):
    a,b: *mixed*
    c: *same*
  right ones ($>$ 2.0):
    d,e: *other*
- By moving the lower bound of the rule's first attribute from -2 to -2.5, the rule would also correctly classify one more example (c). The -2.5 corresponds to the domain value of the last group encountered while going from bottom to top through the *left* unmatched instances with a *same* label (group c).

RG works by finding, for each attribute, new lower and upper bounds by looking for the longest streak of clustered groups with a *same* label within the *left* and *right* unmatched instances respectively. In the *left* ones it searches from bottom to top while in the *right* ones it searches from top to bottom. The new lower bound becomes the domain value of the last group with a *same* label from the search in the *left* unmatched instances, while the new upper bound becomes the domain value of the last group from the search in the *right* unmatched instances. Fig. 5 contains the pseudo-code for this operator.

```
Procedure Rule Generalizing operator
 Input: rule, unmatchedExamplesPerAttributeLeft, unmatchedExamplesPerAttributeRight

maxNumberOfPositiveMatches = 0
Foreach att in numAttributes
    bothLeftAndRight = countLeft = countRight = 0

    pos = getNumClusteredUnmatchesOfAttrLeft(att) - 1
    While getClusteredUnmatchOfAttLeft(att, pos).label == same and pos != -1
        pos --
        countLeft += getClusteredUnmatchOfAttLeft(att,pos).length
    EndWhile
    If countLeft > 0
        bothLeftAndRight = countLeft
        tempPosLeft = pos + 1
    EndIf

    pos = 0
    While getClusteredUnmatchOfAttRight(att, pos).label == same and
          pos < getNumClusteredUnmatchesOfAttrRight(att)
        pos ++
        countRight += getClusteredUnmatchOfAttRight(att,pos).length
    EndWhile
    If countRight > 0
        bothLeftAndRight += countRight
        tempPosRight = pos - 1
    EndIf

    If bothLeftAndRight > maxNumberOfPositiveMatches
        maxNumberOfPositiveMatches = bothLeftAndRight
        Remember tempPosLeft, tempPosRight and att in
                 leftGenPos, rightGenPos and toGen
    EndIf
EndForeach

If maxNumberOfPositiveMatches > 0
    If can generalize left
        rule.lowerBound =
            getClusteredUnmatchOfAttLeft(toGen, leftGenPos).domVal
    EndIf
    If can generalize right
        rule.upperBound =
            getClusteredUnmatchOfAttRight(toGen, rightGenPos).domVal
    EndIf
EndIf

Output: rule
```

Fig. 5: Pseudo-code of the RG operator

4.3 Noise-tolerant memetic operators for continuous attributes

The three continuous-attributes memetic operators have been adapted to deal with noisy-data by altering the criteria they use for modifying the attribute bounds of the classifiers. The noise tolerant operators can edit a rule and make it cover less positive examples, which the normal continuous memetic operators can not do. The percentage of positive examples that can be removed from a rule's cover-set is however, controlled by a parameter: the amount of expected noise of the dataset. For example, if a dataset is estimated to contain 10% of noise (attribute-level or class-level), then the noise-tolerant memetic operators can assume that 10% of the positive examples are actually negative examples and vice-versa. Meaning, for example, for the noise-tolerant RC operator, that it can remove an entire portion of a rule's cover-set if it contains a ratio of 9:1 negative to positive examples.

4.3.1 Alterations to the preprocessing steps

These operators, as the standard continuous-attributes ones, use two preprocessing steps: the M&U operator and the CDS operator. The M&U operator computes, for each rule, lists of matched and unmatched instances sorted on the domain value of each attribute. The mode of operation of this algorithm remains unchanged. However, the CDS operator needed

to be modified. The original CDS operator clusters the output of the M&U operator building groups of instances with the same domain value and labelling them according to the individual instances' class mixture. For the noise-tolerant LS operators to build ratios of positive to negative examples they must be able to count the exact number of positive and negative matched and unmatched instances of each rule. The original CDS operator only maintains the total length for each built group, but not the number of positive and negative instances within the group. As such, for the noise-tolerant local search operators to function, the CDS operator has been updated to keep track of the number of positive and negative instances within each clustered domain segment. As before, when local search is applied to an individual rule, first the M&U operator is applied to the current set of instances (and implicitly to the current rule) to generate an intermediary result, which is then sent through the CDS operator to produce a list of clustered domain segments for each expressed attribute of the rule, which is finally passed as input to the active memetic operators (as before, along with the actual rule to edit).

The mode of operation of the three noise-tolerant continuous-attributes memetic operators are described in detail next.

4.3.2 Noise-tolerant Rule Cleaning

The noise-tolerant RC operator works in the same way as the standard RC, that is, by shrinking the interval of one of the attributes associated to the rule being edited. The only difference is the acceptance criteria of each of the candidate shrinking operations, as follows:

A "clean" operation is characterised by:

- the number of matched positive examples - $\#positive$
- the number of matched negative examples - $\#negative$

And it is considered **valid** if the following holds:

$$\frac{\#negative}{\#positive + \#negative} \geq 1.0 - expectedNoiseAmount$$

This means that a clean application is only valid when the percentage of negative examples that will be removed is at least equal to the percentage of expected non-noisy examples. As such, when the expected amount of noise parameter increases, the number of negatively matched examples that can be removed also increases. The number of matched positive and negative examples are computed by counting the individual number of positive and negative matches contained in the clustered domain segments built by the CDS operator in the final pre-processing stage.

The following example illustrates how the noise-tolerant RC local search operates:

- Given the following rule: $[-2.0, 2.0], [-2.0, 2.0] \mapsto 0$
- In a dataset with an expected noise amount of 33%
- And the matched instances of the rule sorted on the first attribute (not also clustered):
  a: -1.0, 1.5 || 1
  b: 0.1, 0.6 || 0
  c: 0.1, 0.7 || 1
  d: 1.2, 1.6 || 0
  e: 1.7, 0.4 || 0
- The rule classifies correctly three examples and incorrectly two (a and c).
- Clustering (while also counting the number of positive and negative matches between parentheses) the sorted matched instances we get:
  a: *other (0,1)*
  b,c: *mixed (1,1)*
  d: *same (1,0)*
  e: *same (1,0)*
- By shifting the lower bound of the first attribute's interval of the rule from -2 to 1.2 the rule will classify correctly two examples and none incorrectly. The 1.2 is the domain value of the first group (in this case d) which represents the valid clean containing the highest number of examples to be cleaned found by traversing the clustered domain segments from top to bottom. The clean validity condition in this case would evaluate to *true*: $2/3 \geq 1 - 0.33 \Leftrightarrow 0.66 \geq 0.66$.

Thus the noise-tolerant rule cleaning procedure finds, for each attribute, the smallest interval which enables the rule to cover the lowest number of misclassified examples while maintaining a sensitive number of correctly classified examples within the rule's cover set. It does this by going through the clustered sorted matches from the top to the bottom (for the lower bound and in reverse for the upper bound) and maintaining two counters for the number of misclassified and correctly classified examples. It then computes the validity of each cleaning position, only remembering the valid clean possibilities with the highest total number of examples removed. Then new lower and/or upper bounds are built using the domain values of the best of all these clean possibilities.

The following formula for assigning a score to a clean possibility has been used to define a comparison relation between two clean possibilities (to be able to pick the best clean option over all attributes):

$$cleanScore = \frac{\#negative}{\#positive + \#negative} * \sqrt{\#negative}$$

This formula is based on the clean validity definition but also weights-in the number of negative examples preferring rule clean options which remove greater numbers of negative examples. The square root of the number of negative examples is taken to provide a balance between rule clean possibilities with a high percentage of misclassified examples removed but which remove a relatively low number of such examples

and those with a lower percentage of misclassified examples removed but which remove a relatively high number of such examples.

Only the pair of bounds corresponding to the rule clean options which have the highest score, according to the formula defined above, are actually substituted with their respective counterparts; meaning that only one clean operation is performed. Fig. 6 contains the pseudo-code for this operator.

```
Procedure Noise-tolerant Rule Cleaning operator
Input: rule, matchedExamplesPerAttribute

maxScore = 0
Foreach att in numAttributes
    bestLToRClean = bestRToLClean = null
    numMatches = (0,0)

    For pos = 0, pos < getNumClusteredMatchesOfAtt(att), pos ++
        Maintain numMatches by counting positive and negative examples
        If numMatches represents a valid clean
            Set bestLToRClean to current clean
        EndIf
    EndFor

    Reset numMatches

    For pos = getNumClusteredMatchesOfAtt(att), pos > 1, pos --
        Maintain numMatches by counting positive and negative examples
        If numMatches represents a valid clean
            Set bestRToLClean to current clean
        EndIf
    EndFor

    bestScoreForThisAttr = compute combination of bestLToRClean and
                           bestRToLClean with best score

    If bestScoreForThisAttr > maxScore and clean boundaries are not in reverse
        maxScore = bestScoreForThisAttr
        Remember best clean application in leftCPos, rightCPos and toClean
    EndIf
EndForeach

If maxScore > 0
    If can clean left
       rule.lowerBound = getClusteredMatchOfAtt(toClean, leftCPos).domVal
    EndIf
    If can clean right
       rule.upperBound = getClusteredMatchOfAtt(toClean, rightCPos).domVal
    EndIf
EndIf

Output: rule
```

Fig. 6: Pseudo-code of the noise-tolerant RC operator

4.3.3 Noise-tolerant Rule Splitting

The noise-tolerant splitting operator applies the standard RS to find the optimal domain segment to remove from the rule to make it drop the maximum number of negative examples without affecting the positively matched ones. Afterwards, the noise-tolerant RS extends this domain segment using the same relaxed validity criteria used by the noise-tolerant RC. Conceptually, this operator simply applies the noise-tolerant RC to the subrules generated by the standard RS. The mode of operation of the noise-tolerant RS operator is shown through the following example:

- Given the following rule: $[-2.0, 2.0], [-2.0, 2.0] \mapsto 0$
- In a dataset with an expected noise amount of 33%
- And the matched instances of the rule sorted on the first attribute (not also clustered):
  a: -0.7, 0.2 $\|$ 1
  b: 0.5, 1.8 $\|$ 0
  c: 1.1, 1.2 $\|$ 1
  d: 1.1, 0.0 $\|$ 1
  e: 1.8, 0.3 $\|$ 1
  f: 2.0, 1.5 $\|$ 0
- The rule classifies correctly two examples and incorrectly four (a, c, d and e).
- Clustering the sorted matched instances (and maintaining negative and positive matched instances counters) we get:
  a: *other (0,1)*
  b: *same (1,0)*
  c,d: *other (0, 2)*
  e: *other (0.1)*
  f: *same (1,0)*
- The standard RS operator would eliminate the consecutive misclassified examples by splitting the original rule into two subrules on the interval of the first attribute and omitting the (0.5, 2.0) section of the interval which covers the misclassified instances c, d and e. Then, one subrule would cover examples a and b, having the interval of the first attribute set to [-2.0, 0.5], while the other one would cover example f with the interval of the first attribute set to [2.0, 2.0]. Collectively, the two subrules would correctly classify the same number of examples (2) but misclassify only one example, instead of four.
- The extra step performed by the noise-tolerant RS operator would be to also extend the interval of only negative examples from its original (0.5, 2.0) to (-0.7, 2.0). This interval would remove four negative and one positive example, constituting a valid extension (applying the noise-tolerant RC validity condition) since: $4/5 \geq 1 - 0.33 \Leftrightarrow 0.80 \geq 0.66 \Leftrightarrow true$.

Fig. 7 contains the pseudo-code for this operator.

4.3.4 Noise-tolerant Rule Generalizing

Similarly to the noise-tolerant RC and RS versions, the noise-tolerant rule generalizing operator works as the standard RG except for using a relaxed generalize operation validity criteria (based on the ratio between the number of matched positive examples and the total number of matched examples), as exemplified below:

- Given the following rule: $[-2.0, 2.0], [-2.0, 2.0] \mapsto 0$
- In a dataset with an expected noise amount of 33%

```
Procedure Noise-tolerant Rule Splitting operator
Input: rule, matchedExamplesPerAttribute

maxScore = 0
bestSplitAtt = -1
bestLeftPos = bestRightPos = -1

Foreach att in numAttributes

    Run standard RS to try and find longest streak of negative matches for att
    Store result in startCleanPos, endCleanPos and attToSplit

    If attToSplit == att
        numMatches = (0,0)
        bestLeftClean = bestRightClean = null

        For pos = startCleanPos - 1, pos > 1, pos --
            Maintain numMatches by counting positive and negative examples
            If numMatches represents a valid interval extension
                Set bestLeftClean to current extension
            EndIf
        EndFor

        Reset numMatches

        end = getNumClusteredMatchesOfAttr(att)
        For pos = endCleanPos+1, pos < end, pos ++
            Maintain numMatches by counting positive and negative examples
            If numMatches represents a valid interval extension
                Set bestRightClean to current extension
            EndIf
        EndFor

        bestScoreForThisAttr = compute combination of bestLeftClean and
                               bestRightClean with best score

        If bestScoreForThisAttr > maxScore
            maxScore = bestScoreForThisAttr
            Remember best split in bestLPos, bestRPos and bestSplitAtt
        EndIf

    EndIf

EndForeach

If maxScore > 0

    Create two subrules as copies of rule: r1, r2

    r1.upperBound = getClusteredMatchOfAtt(bestSplitAtt, bestLPos - 1).domVal

    r2.lowerBound = getClusteredMatchOfAtt(bestSplitAtt, bestRPos + 1).domVal

    Output: r1, r2

Else

    Output: rule

EndIf
```

Fig. 7: Pseudo-code of the noise-tolerant RS operator

- The unmatched instances which only failed to be matched by the first attribute are computed and sorted, for example:
  left ones ($<$ -2.0):
  a: -4.7, 1.3 $\|$ 0
  b: -3.5, 0.9 $\|$ 1
  c: -3.5, -1.5 $\|$ 0
  d: -2.5, 1.2 $\|$ 0
  right ones ($>$ 2.0):
  e: 3.4, 1.8 $\|$ 1
  f: 3.4, 1.8 $\|$ 1
- Then clustered (while also counting positive and negative examples):
  left ones ($<$ -2.0):
  a: *same (1,0)*
  b,c: *mixed (1,1)*
  d: *same (1,0)*
  right ones ($>$ 2.0):
  e,f: *other (0,2)*
- By moving the lower bound of the rule's first attribute from -2 to -4.7, the rule would correctly classify 3 more examples (a, c, d) but would also match a new negative example (b). The -4.7 corresponds to the domain value of the last group which represents the valid generalize operation containing the largest number of examples to be added to the rule's cover set found by traversing the *left* clustered domain segments from bottom to top.

Fig. 8 shows the pseudo-code of this operator.

```
Procedure Noise-tolerant Rule Generalizing operator
Input: rule, unmatchedExamplesPerAttrLeft, unmatchedExamplesPerAttrRight

maxScore = 0
Foreach att in numAttributes
    bestLToRGen = bestRToLGen = null
    numMatches = (0,0)

    For pos = getNumClusteredUnmatchesOfAttrLeft(att), pos ≥ 0, pos --
        Maintain numMatches by counting positive and negative unmatches
        If numMatches represents a valid generalize
            Set bestLToRGen to current generalize operation
        EndIf
    EndFor

    Reset numMatches

    For pos = 0, pos < getNumClusteredUnmatchesOfAttrRight(att), pos ++
        Maintain numMatches by counting positive and negative unmatches
        If numMatches represents a valid generalize
            Set bestRToLGen to current generalize operation
        EndIf
    EndFor

    bestScoreForThisAttr = compute combination of bestLToRGen and
                           bestRToLGen with best score

    If bestScoreForThisAttr > maxScore
        maxScore = bestScoreForThisAttr
        Store best generalize operation in leftGenPos, rightGenPos and toGen
    EndIf
EndForeach

If maxScore > 0

    If can generalize left

        rule.lowerBound =

            getClusteredUnmatchOfAttLeft(toGen, leftGenPos).domVal

    EndIf

    If can generalize right

        rule.upperBound =

            getClusteredUnmatchOfAttRight(toGen, rightGenPos).domVal

    EndIf

EndIf

Output: rule
```

Fig. 8: Pseudo-code of the noise-tolerant RG operator

5 Experimental design

This section presents the experimental protocol we have followed to evaluate the memetic extensions of BioHEL presented in this paper. Four stages of experiments have been performed. The abbreviation ESn, where n is the number of the stage, has been used throughout the text to refer to individual experiments stages. The first three stages use synthetic datasets, where the structure of each problem is known and can be used when analysing the experiments' results. The fourth, validation, stage uses a combination of synthetic and real-world datasets. For each experiment ten repetitions with different random seeds have been performed for each configuration. All experiments have been run on uniform hardware, namely the High Performance Computing Facility of the University of Nottingham, using Intel Xeon E5472 processors running at 3.0GHz and the Linux operating system. To analyse the results of the experiments we have used the Friedman test for multiple comparisons. Specifically, we have used this test to determine if there are significant performance differences (using various metrics depending on the experiment: accuracy, run-time or rule-set size) between the tested methods, followed by the Holm post-hoc test to assess the differences between the top method and the rest of the tested configurations, as suggested in [11, 19].

5.1 Datasets

A broad range of datasets (synthetic discrete datasets, synthetic continuous datasets and real-world datasets) have been used in the paper, where each of the different evaluation stages of the paper uses a subset of them chosen for specific reasons. As the test suites of the evaluation stages overlap, we describe all of them here. Table 1 shows an overview of the features of the datasets used in the experiments of this paper. All datasets can be downloaded from `http://cruncher.cs.nott.ac.uk/datasetsMemeticBioHEL.tar.bz2` (19.5GB).

5.1.1 Synthetic datasets with discrete attributes

- *20-bit multiplexer:* An n-bit multiplexer dataset is built by enumerating all possible inputs for the multiplexer function and computing their associated class values. The multiplexer function is defined on bit strings of length $2^k + k$ and its output is given by computing the decimal interpretation of the first 2^k bits and then indexing into the remaining k bits (counting from 0 onwards). The number of instances of each n-bit multiplexer dataset is equal to the number of all possible bit strings of length n which is: 2^n. The 20-bit multiplexer datasets corresponds to a k value of 4. In the results tables this dataset's name is abbreviated as MX20.

- *Hybrid multiplexer-parity dataset:* The hybrid multiplexer-parity dataset [9] used here is an encoding of the 6-bit multiplexer dataset using a 3-parity function. For brevity it is referred to as the *ParMX* dataset. To obtain the original multiplexer dataset from this dataset, the 3-parity function must be applied to the entire bit string of each encoded instance. The 3-parity function takes as input 3 bits and outputs 0 if their sum is even and 1 otherwise. The dataset is composed from 2^{18} instances and can be optimally classified by 256 rules, excluding the default rule which covers the remaining instances.

- *Discrete k-DNF datasets:* These parametrised datasets have been widely used within the Computational Learning Theory field [24]. A k-DNF dataset instance consists of the pairs of all possible inputs and computed outputs of a boolean function defined as a disjunctive normal form (DNF) predicate. The predicate is formed from n disjunctive terms, corresponding to the rules that need to be learned. Each term is the conjunction of a randomly chosen subset k out of the d possible attributes of the domain. All instances that make the DNF predicate true will have class 1, and all others class 0.

5.1.2 Synthetic datasets with continuous attributes

- *Continuous k-DNF datasets:* The equivalent continuous k-DNF datasets have been devised by using attributes with continuous domains, where each term of each rule is a set membership condition defined on a continuous half of the $[0.0, 1.0]$ interval. The bounds of each interval are constructed by selecting a value in $[0.0, 0.5]$ with uniform probability for the lower bound and then creating the upper bound by adding 0.5 to the lower one. An example of a full rule of a continuous k-DNF is shown below:
  $Rule : x1 \in [0.34, 0.8] \wedge x3 \in [0.03, 0.53] \wedge x0 \in [0.4, 0.54] \rightarrow$ class 1
  To generate a complete dataset, n rules are constructed. For each rule, k attributes are sampled without replacement from the d possible ones and the bounds for the expressed attributes are generated using the method specified above. To generate the examples of the dataset, values for all d attributes are randomly picked with uniform probability from $[0.0, 1.0]$ and then matched against each rule. The class of the instance is set to 1 (corresponding to the *true* logical value) if the instance is matched by at least one rule in the k-DNF and to 0 (corresponding to the *false* logical value) otherwise. The number of instances in the continuous k-DNF is not determined by the (k, d, n) parametrisation tuple and can be set at de-

sired values to influence the size and difficulty of the problem. For this paper we have generated as many instances for each continuous k-DNF dataset as its equivalent discrete problem has. In the results tables these datasets are abbreviated as ckDNF20-k$< N >$-d20, where $< N >$ indicates the number of attributes sampled from the 20 possible ones, where each such dataset is comprises from 20 rules.

– *Checkerboard datasets:* The Checkerboard dataset is a simple 2D 4x4 pattern of alternating black and white tiles where each tile ocupies 25% of the attribute's domain. Moreover, up to 18 extra irrelevant attributes are added to the problem producing a family of 19 datasets. For each dataset 1000 instances are sampled (with uniform probability for the two attributes that define the checkerboard pattern, and with a gaussian distribution centered at 50% of the attribute's domain for the irrelevant features). The class of each instance corresponds to the colour of the tile sampled by the instance's two relevant attributes.

5.1.3 Real-world datasets

Real-world datasets are used in the final validation stage of the paper. We have used ten datasets of medium/large size. Five of them (Adult, connect-4, pen-based character recognition (pen), waveform (wav) and kddcup99) come from the University of California at Irvine (UCI) repository [18]. The other five (SS1 and CN-W1, CN-W2, CN-W3 and CN-W4) come from our own repository of bioinformatics datasets, and we have employed them in the past to evaluate our developments in evolutionary learning methods [6, 14, 16]. Specifically, CN-W1, CN-W2, CN-W3 and CN-W4 are four different versions of the same dataset: All datasets are trying to learn the same problem, but each version using a different degree of resolution in its representation which corresponds to a number of attributes ranging from 60 (in W1) to 180 (in W4). With these four versions of the same dataset we are aiming to test the scalability of our method in relation to the number of attributes in the problem.

Table 1 Features of the datasets used in this paper. #Inst. = Number of Instances, #Attr. = Number of attributes, #Disc. = Number of discrete attributes, #Cont. = Number of continuous attributes, Dev.cla. = Standard deviation of the set of class counts

Name	#Inst.	#Attr.	#Disc.	#Cont.	Dev.cla.
Synthetic datasets with discrete attributes					
ParMX	262144	18	18	0	0.00%
MX20	1048576	20	20	0	0.00%
kDNF20-k5-r20	1048576	20	20	0	4.42%
kDNF20-k6-r20	1048576	20	20	0	33.08%
kDNF20-k7-r20	1048576	20	20	0	50.05%
kDNF20-k8-r20	1048576	20	20	0	59.96%
kDNF20-k9-r20	1048576	20	20	0	65.27%
kDNF20-k10-r20	1048576	20	20	0	67.97%
Synthetic datasets with continuous attributes					
ckDNF20-k5-r20	1048576	20	0	20	10.14%
ckDNF20-k6-r20	1048576	20	0	20	35.57%
ckDNF20-k7-r20	1048576	20	0	20	50.99%
ckDNF20-k8-r20	1048576	20	0	20	60.56%
ckDNF20-k9-r20	1048576	20	0	20	65.36%
ckDNF20-k10-r20	1048576	20	0	20	67.94%
Checkerboard datasets	1000	2..20	0	2..20	0.00%
Real-world datasets					
wav	5000	40	0	40	0.4%
pen	10992	16	0	16	0.4%
Adult	48842	14	8	6	26.1%
connect-4	67557	42	42	0	23.8%
SS1	83823	300	0	300	8.3%
CN-w1	257560	60	0	60	0.0%
CN-w2	257560	100	0	100	0.0%
CN-w3	257560	140	0	140	0.0%
CN-w4	257560	180	0	180	0.0%
kddcup	494020	41	15	26	12.6%

5.2 Experiments Stage 1: Checkerboard variants

The first stage of experiments consists of a large-scale evaluation of 35 configurations of memetic search on the 19 variants of the Checkerboard dataset. The objective of this stage is to identify which configurations of operators improve BioHEL's learning capacity and which only add useless computational effort in a small dataset. Furthermore, because in the Checkerboard only two attributes of the domain are relevant, this stage also evaluates how the memetic operators influence BioHEL's ability to identify the correct attributes for each dataset and avoid overfitting the problem, especially in the dataset variants with a large number of irrelevant attributes.

The 35 different memetic search configurations result from five separate individual-wise probabilities of applying the memetic operators along with seven modes of combining the three rule-wise memetic operators. The tested individual-wise probabilities are: 5%, 10%, 15%, 20% and 25%, while the combinations of the memetic operators used consist of testing each operator individually, then all the groups of two different operators and finally applying all three operators. Enumerated, the seven combinations of memetic operators are: $\{RC\}$, $\{RS\}$, $\{RG\}$, $\{RC,RS\}$, $\{RC,RG\}$, $\{RS,RG\}$ and $\{RC,RS,RG\}$. Besides the 35 LS configurations, a baseline configuration consisting of the basic BioHEL system has been included. In this stage of experiments ten-fold cross-validation has been used to assess the generalisation potential of BioHEL's memetic extensions. In ES1 accuracy and runtime are used as performance metrics.

5.3 Experiments Stage 2: Local comparison

The second stage of experiments is targeted on the same set of 35 local search configurations, but tackling a more diverse group of datasets, with both continuous and discrete attributes. The purpose of this stage is to carry out an initial assessment of the performance of the memetic configurations in terms of accuracy and runtime on large datasets.

Due to the potentially high computational cost of these experiments and the fact that all datasets are synthetic and we know the optimal solution for each problem we will not evaluate the accuracy of these methods based on ten-fold cross-validation. Instead, we will follow the same procedure we employed in [6] and just use training accuracy and run the system for as many GA iterations as necessary to obtain the perfect solution to the problem. In this way we can assess the exact amount of computational effort that each of the 36 tested settings (35 memetic operators and the basic BioHEL) requires to generate the optimal solution.

Given that BioHEL evolves rules and not rule sets we cannot know throughout the GA iterations if the full solution to the problem has been generated, but we can know if we have obtained any of the optimal rules that form such solution. To do so we draft what we call the specificity-based GA iteration stop condition as follows: the GA cycle will stop when (a) a rule with 100% accuracy has been obtained **and** (b) for discrete problems that rule has a number of specified attributes that correspond to the optimal rules of the tested problem, for continuous problems, the volume v_r specified by the rule's hyperrectangle is approximately equal to the volume of one of the optimal rules v_o, where $v_r \approx v_o \Leftrightarrow |v_r - v_o|/max(v_r, v_o) \leq 0.05$. The datasets employed for evaluation are the following: MX20, ParMX and the discrete and continuous k-DNF datasets with $d = 20$, $n = 20$ and $k \in \{5,6,7,8,9,10\}$. In this stage, runtime and accuracy rankings, average accuracy on the training set, average runtime and average number of rules in the final solution have been used as performance metrics.

5.4 Experiments Stage 3: Noise-tolerant operators

The third stage applies the top memetic configurations identifed in the two previous stages to 9 noisy variants of the ckDNF20-k5-r20 dataset. These nine variants have been produced by adding three types of noise with three different probabilities to the original dataset. The three types of noise that have been used are: instance class flipping, instance domain value alteration and applying both types of noise at once. The three probabilities of applying noise used are: 5%, 15% and 25%. In the case of flipping the class, a 5% noise level simply means that ∼5% of all instances in the dataset have had their class value flipped from a 0 to a 1 or in reverse. In the case of modifying the domain values of instances, a 15% noise level means that ∼15% of all domain values in all instances in the dataset have had uniform noise in $[-0.1, 0.1]$ added. In ES3 ten-fold cross-validation is used.

One purpose of ES3 is to show that BioHEL's performance on noisy datasets is improved when using the noise-tolerant memetic extensions when compared to the baseline system. Another goal is to quantify the difference between the noise-tolerant and the standard memetic operators in terms of runtime complexity and/or solution accuracy, and to identify an applicability criteria for both the noise-tolerant and the standard LS operators based on the noise level and type of a dataset. In ES3 the average accuracy, average runtime and average number of rules in the final solution have been used as performance metrics.

5.5 Experiments Stage 4: Global comparison

The fourth and final stage of experiments consists of a comparison of BioHEL's memetic operators with other machine learning algorithms. We have selected four different algorithms that have rule/tree-based knowledge representations, which are essentially equivalent in expressive power to BioHEL's representation, in order to have a maximally fair comparison. The chosen algorithms are the GAssist [1] evolutionary learning algorithm (using our own C++ implementation) and the WEKA [21] implementations of C4.5, RandomForest (RF) and PART. There are many evolutionary learning algorithms reported in the literature that have reported competent performance against other machine learning methods [13]. However, given that the test suite of this paper mostly includes datasets with large sets of instances, we have selected only GAssist because its code base is adapted to handle such large training set sizes.

From BioHEL we are including its basic version and the three best memetic configurations identified in ES1 and ES2. For this final stage of experiments we are employing stratified ten-fold cross-validation and test accuracy instead of the specificity-based stop condition and training accuracy employed in ES2. Default settings have been used for all machine learning methods in the comparison (specifically, the default parameters in WEKA 3.6.6 have been used) except for GAssist, where two parameters have been altered from its default values [1]: the number of GA iterations has been set up to 10000 and the number of strata in ILAS[1] has been set up to 50.

The datasets chosen for this final stage of experiments are a mix of synthetic and real-world datasets. The included synthetic datasets are MX20, ParMX, all discrete k-DNF datasets and the continuous k-DNF datasets with $k \in \{5,6,7\}$. All real-world datasets described in the previous section have been included. The goal of this stage is to compare the results, in terms of accuracy and rule set size of the solutions, obtained by BioHEL's memetic extensions with the results of other state-of-the-art machine learning methods. Given that both BioHEL and GAssist are implemented in C++ and WEKA is implemented in Java, a runtime comparison is not included as this would only be approximate at most.

[1] briefly described in the next subsection

5.6 BioHEL configuration

The basic BioHEL configuration is shown in table 2 while the dataset-specific parameters are shown in table 3. The specificity parameter of table 3 shows the parameter to be passed to the specificity-based GA iteration stop condition; for the continuous-attributes datasets it shows the optimal classifier hyper-volume and for the discrete-attributes datasets it presents the optimal number of expressed attributes in a GABIL rule. Three parameters of BioHEL have been adjusted for each dataset: the default class policy, the coverage breakpoint and the number of windows in the ILAS windowing scheme.

The default class policy has been changed from its default value (choosing the majority class as default) on the k-DNF datasets so it always maps to the 0 class. The reason is that in these problems the rules in the solution only map to class 1 so the problem already has an implicit default class. With a wrong default rule we would not be able to employ the specificity-based stop condition and hence we would not be able to assess the performance of the tested memetic operators with the same level of certainty. Moreover, the coverage breakpoint is the parameter from BioHEL's fitness function that regulates the balance between accuracy and coverage in the evolved rules. We have set it up to the optimal value for each dataset in order to avoid adding an extra layer of difficulty to the learning process, as in this study we are only concerned with observing the effect of the memetic extensions. A thorough study of the impact of both the default class and the coverage breakpoint parameter in kDNF datasets is available elsewhere [15]. Finally, ILAS is an efficiency enhancement mechanism [3] that partitions the training set into strata, and uses a different stratum in each GA iteration in a round-robin fashion. The number of strata employed has been set up to the maximum value possible that does not create an impact in the learning capacity of BioHEL. On the real-world datasets, the performance of GAssist and BioHEL has been enhanced by using a simple consensus ensemble mechanism [4], in which the predictions for each test set are performed by a majority vote of an ensemble of rule sets learnt from the repetitions with different random seeds of the rule learning process from the corresponding training set. This mechanism has shown to improve the accuracy of GAssist and BioHEL in a broad range of scenarios [8,17,7]. Finally, for all the experiments with real-world datasets we have employed the noise-tolerant versions of the memetic operators.

For the full explanation of BioHEL's configuration parameters please see [2].

Table 2 BioHEL baseline parameters

Parameter	Value
General parameters	
Iterations	50
Crossover probability	0.6
Selection algorithm	tournament
Tournament size	4
Population size	500
Individual wise mutation probability	0.6
Repetitions of rule learning process	2
MDL-based fitness function	
Iteration of activation	10
Initial theory length ratio	0.25
Weight relax factor	0.90
Coverage ratio	0.90
Representation parameters	
Probability of 1 in GABIL representation	0.75
Expected value of #expressed att. in init.	15
Probability of generalize in ALKR	0.10
Probability of specialize in ALKR	0.10

Table 3 Dataset specific parameters

Dataset	Def. class	Specificity	#strata
MX20	major	5	2500
ParMX	major	9	40
All Checkerboards	major	0.0625	8
Discrete k-DNFs	0	k	50
Continuous k-DNFs	0	0.5^k	50
Real-world datasets	major	0.005	50

6 Results and analysis

Next we present the results obtained in the four evaluation stages.

6.1 Results for Experiments Stage 1: Checkerboard variants

In this subsection we show a summary of the results of evaluating 35 memetic configurations on all 19 variants of the Checkerboard dataset.

First, we will analyse the accuracy and runtime performance of the different combinations of local search operators and probabilties of application. In figure 9 and table 4 the results of the Friedman tests and the post-hoc Holm tests are shown. Both visualisations show the accuracy and runtime ranks of all methods tested, averaged across all datasets. The full details of the results are reported in the appendix of the paper, in tables 16 and 17.

The Friedman tests have been run separately on both testing accuracy and runtime data from all 19 datasets, with the returned p-values of both tests being less than $2.2e-16$, indicating significant performance differences. The post-hoc Holm tests have been performed by comparing the accuracy and runtime ranking of the top method with all other Bio-

HEL configurations. The significant differences at 99% and 95% significance levels are shown using dotted horizontal lines in table 4. All configurations listed below the first line have a significantly worse performance (at 95% significance level) than that of the top ranked method.

From this table we can observe that in terms of runtime, the Basic configuration and the top eight memetic configurations are similar in performance. However, RG alone appears to not be beneficial in reducing the runtime requirements for the Checkerboard variants as it does not appear in the top eight runtime rankings. On the other hand, RS, either alone or in groups with RC or RG, appears to have the most impact in reducing BioHEL's runtime in these small datasets, as it appears in the majority of the top runtime rankings. We can also observe that, in general, higher individual-wise probabilities of applying the memetic operators result in longer runtimes.

Table 4 ES1: Results of Friedman-Holm statistical tests on ES1 runtime and accuracy data. Avg.R. = Average rank of runtime or accuracy respectively

Runtime		Test Accuracy	
Config.	Avg.R.	Config.	Avg.R.
0.05-RS	1.16	0.15-RS-RG	8.00
0.05-RS-RG	3.16	0.10-RS-RG	8.84
0.05-RC-RS	4.79	0.05-RS-RG	9.11
0.05-RC-RS-RG	5.00	0.20-RS-RG	9.47
0.10-RS	5.13	0.25-RS-RG	9.68
Basic	8.21	0.10-RS	10.26
0.05-RC	8.79	0.20-RS	12.58
0.05-RC-RG	8.89	0.20-RC-RS-RG	12.58
0.10-RC-RS	9.32	0.05-RS	12.89
0.15-RS	10.79	0.15-RS	12.89
0.10-RC	12.42	0.05-RC-RS-RG	14.53
0.10-RS-RG	12.42	0.10-RC-RS-RG	15.32
0.10-RC-RS-RG	14.21	0.15-RC-RS-RG	15.32
0.15-RC-RS	14.42	0.25-RS	15.68
0.05-RG	14.97	0.20-RC-RG	15.84
0.15-RC	16.26	0.25-RC-RG	16.00
0.10-RC-RG	16.53	0.15-RC-RS	17.00
0.20-RS	16.63	0.25-RC-RS-RG	17.63
0.20-RC	19.11	0.10-RC-RS	17.95
0.20-RC-RS	20.53	0.15-RC-RG	18.11
0.15-RS-RG	21.16	0.05-RC-RS	18.26
0.25-RC	21.74	0.10-RC-RG	19.53
0.15-RC-RS-RG	22.32	0.20-RC-RS	20.16
0.10-RG	22.79	0.25-RC-RS	21.58
0.25-RS	22.89	0.25-RG	21.74
0.15-RC-RG	24.42	0.25-RC	22.95
0.25-RC-RS	25.89	0.20-RC	23.84
0.20-RS-RG	28.16	0.20-RG	23.84
0.20-RC-RS-RG	28.47	0.05-RC-RG	24.00
0.20-RC-RG	29.00	0.15-RC	25.58
0.15-RG	29.47	0.10-RC	26.16
0.25-RC-RG	32.58	0.10-RG	27.11
0.20-RG	32.63	0.15-RG	28.26
0.25-RC-RS-RG	33.05	0.05-RC	28.32
0.25-RS-RG	33.42	0.05-RG	30.53
0.25-RG	35.26	Basic	34.47

In terms of testing accuracy, all memetic operators have a better mean rank than the Basic configuration. Moreover, RS alone or in groups with the other operators also appears to be beneficial for improving the accuracy of the system, as it appears in all the top 15 best ranking configurations; among these, the RS-RG combination seems to provide the optimal level of specificity and generality pressures for BioHEL to evolve the best rule-sets. A similar performance is also obtained by the individual RS operator and the combination of all three operators. The lowest ranking configurations involve the individual RC and RG operators; this is expected since separately these two operators provide only specificity and generality pressures respectively, while a perfect balance of both types of pressures is required for evolving maximally general and accurate rules.

To be able to better observe the relation between the accuracy and runtime performance of the different memetic configurations tested figure 9 shows the accuracy rankings plotted against the runtime rankings for each configuration. We can observe that the 0.05-RS-RG configuration represents the best trade-off between accuracy and runtime performance. Other good trade-offs include 0.15-RS, 0.10-RS and 0.10-RS-RG. It can also be seen that in terms of both runtime and accuracy rankings the following configurations are all superior to the Basic setting: 0.10-RS, 0.05-RC-RS-RG, 0.05-RC-RS, 0.05-RS-RG and 0.05-RS. As such, the RS and RG local search configurations with a low individual-wise probability comprise the optimal BioHEL settings for tackling these datasets.

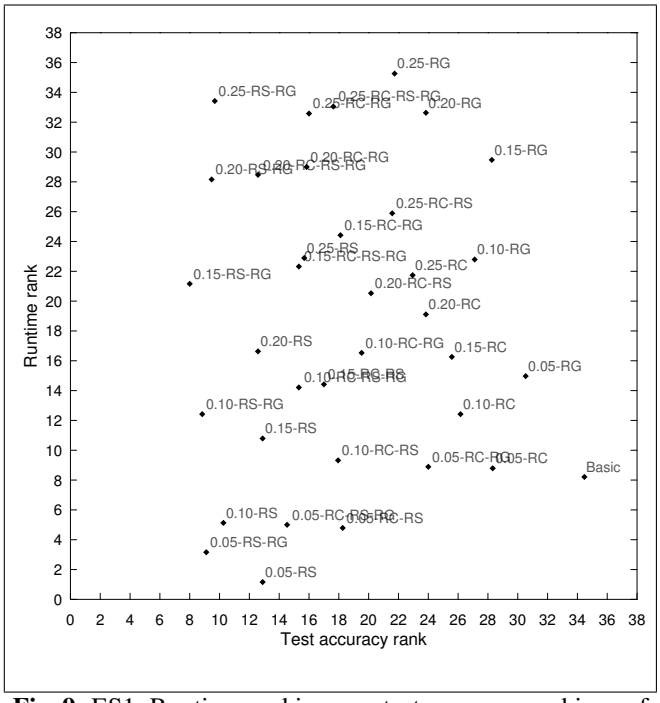


Fig. 9: ES1: Runtime rankings vs. test accuracy rankings of all configurations on all Checkerboard variants

Next, we will investigate the capacity of the LS settings to identify the relevant attributes of a domain. Figure 10 shows the evolution of the test set accuracy plotted against the number of attributes of each Checkerboard variant for the top five memetic configurations and the Basic one. We can notice that all configurations manage to identify the correct attributes of the problem with up to 6 added irrelevant features. Afterwards, the performance of the Basic system starts to degrade while the other methods still obtain good performance up to 13 added irrelevant features. The best accuracy in the last dataset of was obtained by the 0.10-RS-RG combination, with the worst one obtained by the Basic setting.

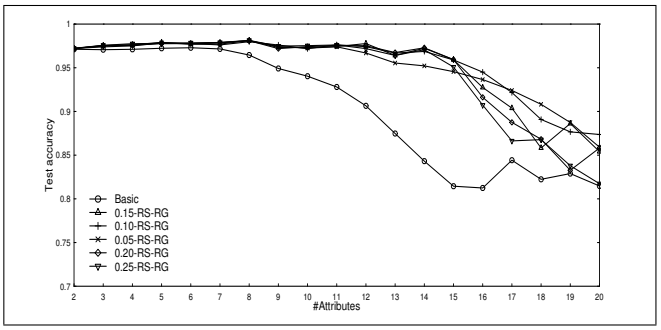


Fig. 10: ES1: Evolution of test accuracy over the number of attributes for the Checkerboard variants showing the top five memetic configurations and the Basic one

6.2 Results for Experiments Stage 2: Local comparison

This subsection presents the results of a comprehensive evaluation of the 35 memetic configurations on the MX20, ParMX and six variants of the continuous and of the discrete k-DNF problems. First we will analyse the runtime and train set accuracy rankings of the LS settings which have been computed using Friedman statistical tests and post-hoc Holm tests on data from all datasets. Table 5 shows the average rankings for runtime (Friedman p-value of $3.97e-05$) and train accuracy (Friedman p-value of $1.351e-05$). The results of the Holm test, with 95% and 99% significance levels are shown through the use of dotted horizontal lines, as in section 6.1. The full details of the results are reported in the appendix of the paper, in tables 18 and 19.

For the accuracy category, the post-hoc Holm test did not discover any significant differences, while for runtime almost all BioHEL configurations were significantly better than the worst setting (0.20-RG) including the Basic one. The reason for these results is that for the majority of datasets all configurations achieved perfect accuracy resulting in many tied rankings. However, if we simply observe the distribution of the memetic operators within the runtime rankings we notice that the configurations where all three operators are acting in unison are in the top part of the table, while RC and RG applied individually are at the bottom of the table, with combinations of RC and RG with RS scattered towards the top part. This would suggest that RS, on its own, but especially when coupled with the other two operators, is beneficial for reducing the overall system runtime. Keeping in mind that the specificity condition is enabled, and because RS generates two accurate sub-rules with a small coverage, these rules with perfect accuracy on the training set most likely stop the GA process early thus causing the runtime reduction.

When compared to the Basic system, the majority of the memetic extensions tested perform better runtime-wise. Performing a similar distribution analysis of the rankings, but now for accuracy, we observe that the superior configurations include the group RC-RG alone and coupled with RS, while the inferior ones mainly consist of applying RS and RC individually. This indicates that the three operators, when applied in unison or without RS, increase BioHEL's training accuracy. We can also observe that higher individual-wise application probabilities generally yield higher training accuracies. Even though the Holm test did not discover significant differences within the accuracy rankings, we can still observe that all except for two LS configurations are superior to the Basic one.

Figure 11 shows the two rankings plotted against each other. From this figure we can clearly identify the Pareto front amongst the tested extensions, which includes: 0.20-RC-RS-RG, 0.25-RC-RS-RG, 0.10-RC-RS-RG and 0.25-RC-RG. As such, the best two configurations of these (0.25-RC-RS-RG, 0.10-RC-RS-RG) will be further evaluated in the next two experimental stages. This stage did not evaluate generalisation capacity by not employing cross-validation due to physical time considerations. Cross-validation performance of the best memetic settings is assessed in the following evaluation stages.

Now we will look at individual per-dataset results and perform an analysis in terms of train set accuracy, runtime and final rule-set (solution) size. Each such results table contains average accuracy, runtime and rule set size data for the Basic system's performance and the top five memetic configurations. This top is computed by stable-sorting the configurations on three levels, first on accuracy, then on runtime and finally on rule set size. Each table also contains a plot of accuracy over runtime for the Basic and the top three LS settings, which were averaged across all runs for each dataset and configuration. For the 20-bit multiplexer case, with results shown in table 6 we can see that there are no accuracy differences visible, but the top local search configurations converge faster to solutions with more compact sizes than the ones evolved by the Basic system. The shown LS combinations manage to obtain rulesets which are close in size

Table 5 ES2: Results of Friedman-Holm statistical test on ES2 runtime and accuracy data. Avg.R. = Average rank of runtime or accuracy respectively

Runtime		Accuracy on training set	
Config.	Avg.R.	Config.	Avg.R.
0.10-RC-RS-RG	11.79	0.20-RC-RG	12.39
0.25-RC-RS	12.79	0.25-RC-RG	12.50
0.20-RC-RS	13.14	0.25-RC-RS-RG	13.25
0.25-RC-RS-RG	13.71	0.20-RC-RS-RG	13.32
0.15-RC-RS-RG	14.50	0.15-RC-RS-RG	14.82
0.15-RC-RS	14.86	0.15-RC-RG	14.96
0.20-RC-RS-RG	15.07	0.10-RC-RS-RG	15.32
0.25-RS	16.07	0.10-RC-RG	15.93
0.10-RC-RS	16.29	0.20-RC-RS	16.32
0.10-RS-RG	16.29	0.20-RG	17.00
0.25-RC-RG	16.36	0.25-RS-RG	17.39
0.20-RS-RG	16.43	0.25-RG	17.39
0.15-RC-RG	16.43	0.25-RC-RS	17.68
0.05-RC-RS-RG	16.50	0.05-RC-RS-RG	17.75
0.05-RC-RS	16.64	0.20-RS-RG	17.82
0.20-RS	16.93	0.15-RS-RG	18.25
0.15-RS	17.57	0.05-RC-RG	18.50
0.05-RC-RG	17.79	0.10-RS-RG	18.96
0.15-RS-RG	17.86	0.05-RC-RS	19.11
0.10-RS	18.29	0.25-RC	19.21
0.15-RC	18.57	0.10-RC-RS	19.25
0.10-RC-RG	18.64	0.25-RS	19.39
0.05-RS	18.71	0.15-RG	19.93
0.05-RS-RG	19.14	0.15-RC-RS	20.04
0.05-RG	19.29	0.10-RG	20.39
0.25-RS-RG	19.29	0.05-RS-RG	20.46
Basic	19.36	0.20-RC	20.46
0.05-RC	19.64	0.20-RS	20.68
0.25-RC	21.43	0.15-RC	21.36
0.20-RC-RG	21.71	0.05-RS	21.46
0.20-RC	22.14	0.10-RS	21.61
0.10-RC	22.36	0.15-RS	22.11
0.10-RG	24.93	0.05-RC	22.21
0.25-RG	25.64	Basic	22.50
0.15-RG	28.57	0.05-RG	22.89
0.20-RG	31.29	0.10-RC	23.36

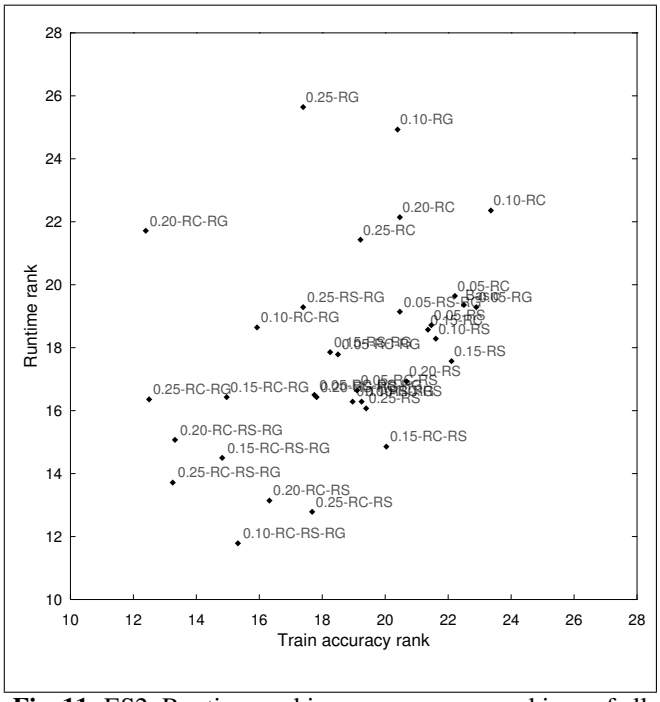
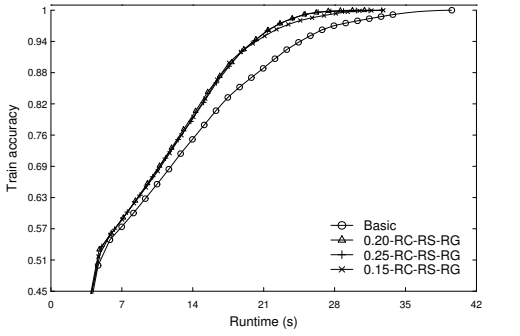


Fig. 11: ES2: Runtime rankings vs. accuracy rankings of all configurations computed across all ES2 datasets

Table 6 ES2: MX20 results

Configuration	Train Accuracy	Runtime (s)	#Rules
Basic	100.00%±0.0%	33.78±5.2	29.00±4.2
0.20-RC-RS-RG	100.00%±0.0%	24.74±3.0	21.90±1.7
0.25-RC-RS-RG	100.00%±0.0%	24.99±3.0	21.50±1.9
0.15-RC-RS-RG	100.00%±0.0%	25.00±5.4	22.30±3.5
0.25-RC-RS	100.00%±0.0%	25.02±3.0	22.80±1.2
0.20-RS-RG	100.00%±0.0%	25.04±2.2	22.80±1.9



to the optimal number of 17 rules for the MX20 dataset (16 rules to classify one class and one default rule for the other class). The plot shows the difference in convergence rate as all memetic configurations shown possess a steeper accuracy increase curve than the Basic system.

The results on the ParMX dataset are presented in table 7. Similarly to MX20, the differences appear at the runtime level, these being more pronounced than in the MX20 case. Morever, the LS settings also produced more compact solutions than the Basic configuration, but even these are too large when compared to the optimal number of 257 rules of this domain.

For all the kDNF datasets tested, the optimal rule set size is composed from 21 rules (20 to cover the instances with class value 1 and one default rule to cover the remaining instances) by their definition ($r = 20$). Tables 8 to 9 show the results on the five discrete kDNF datasets. All 36 configurations of BioHEL were able to evolve the optimal rule set with full training accuracy, with the only differencing factor being the runtime. As k increases, the difference between the runtime of the best LS configurations and the runtime of the Basic system also increases, suggesting that the harder the dataset becomes the more the LS operators can contribute to the efficiency of the learning process. This is also shown in the plots, as the learning curve of the Basic system is consistently below the curves of the best memetic configurations. In the most difficult of these datasets, with higher k values,

Table 7 ES2: ParMX results

Configuration	Train Accuracy	Runtime (s)	#Rules
Basic	99.99%±0.0%	2066.93±351.1	478.20±13.6
0.25-RC-RS	100.00%±0.0%	923.58±105.6	326.20±10.6
0.25-RS	100.00%±0.0%	937.91±146.8	335.60±10.5
0.20-RS	100.00%±0.0%	970.20±130.7	341.70±11.7
0.20-RS-RG	100.00%±0.0%	973.37±73.1	340.80±10.4
0.15-RS	100.00%±0.0%	988.64±43.2	354.10±7.3

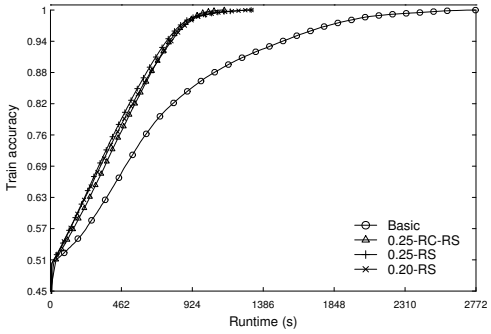
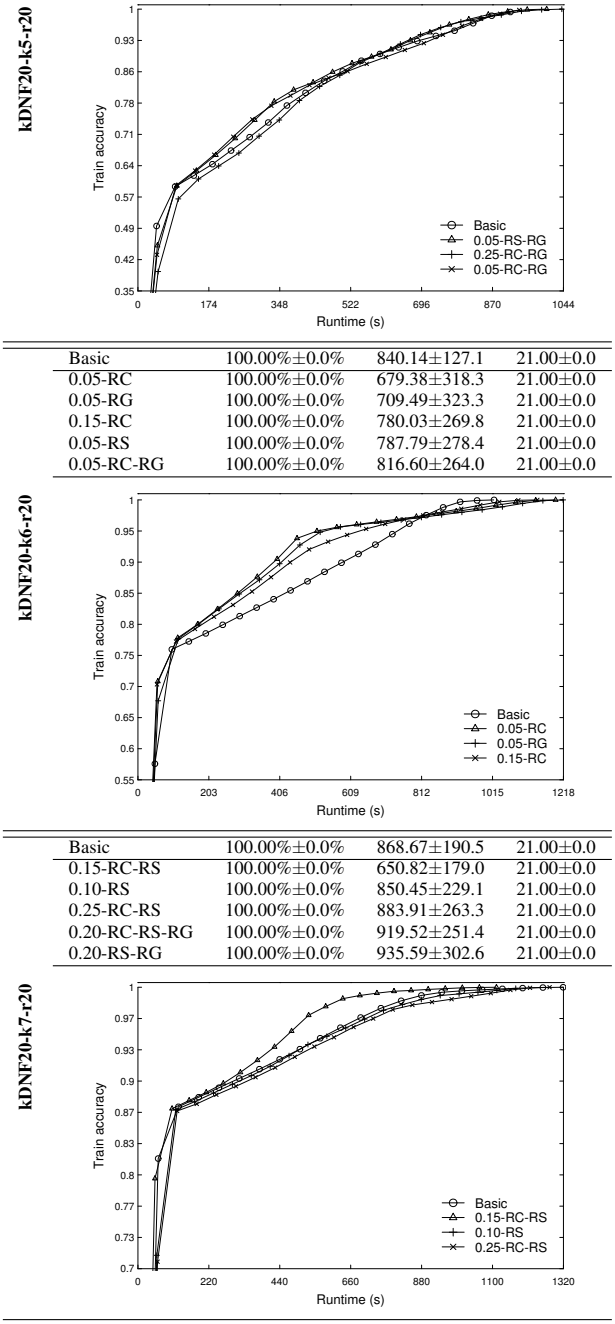


Table 8 ES2: kDNF20-k{5,6,7}-r20 results

D.	Configuration	Train Accuracy	Runtime (s)	#Rules
kDNF20-k5-r20	Basic	100.00%±0.0%	692.31±232.8	21.00±0.0
	0.05-RS-RG	100.00%±0.0%	663.15±262.4	21.00±0.0
	0.25-RC-RG	100.00%±0.0%	668.04±223.1	21.00±0.0
	0.05-RC-RG	100.00%±0.0%	677.96±301.6	21.00±0.0
	0.15-RC	100.00%±0.0%	691.39±275.3	21.00±0.0
	0.25-RS	100.00%±0.0%	692.15±238.8	21.00±0.0
kDNF20-k6-r20	Basic	100.00%±0.0%	840.14±127.1	21.00±0.0
	0.05-RC	100.00%±0.0%	679.38±318.3	21.00±0.0
	0.05-RG	100.00%±0.0%	709.49±323.3	21.00±0.0
	0.15-RC	100.00%±0.0%	780.03±269.8	21.00±0.0
	0.05-RS	100.00%±0.0%	787.79±278.4	21.00±0.0
	0.05-RC-RG	100.00%±0.0%	816.60±264.0	21.00±0.0
kDNF20-k7-r20	Basic	100.00%±0.0%	868.67±190.5	21.00±0.0
	0.15-RC-RS	100.00%±0.0%	650.82±179.0	21.00±0.0
	0.10-RS	100.00%±0.0%	850.45±229.1	21.00±0.0
	0.25-RC-RS	100.00%±0.0%	883.91±263.3	21.00±0.0
	0.20-RC-RS-RG	100.00%±0.0%	919.52±251.4	21.00±0.0
	0.20-RS-RG	100.00%±0.0%	935.59±302.6	21.00±0.0



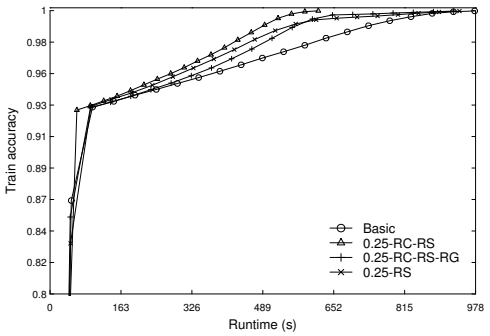
the RC-RS settings consistently appear among the best settings.

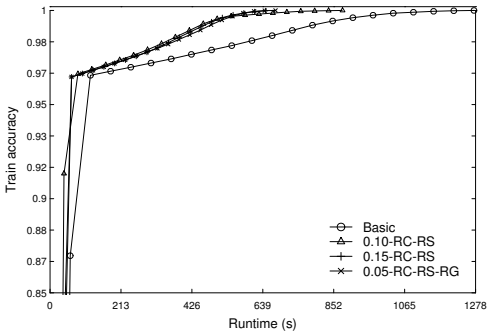
The results on the continuous kDNF problems are presented in tables 10-11. For $k = \{5,6\}$ we can immediately see that the memetic configurations manage to evolve solutions with a near-optimal size, of around 22, while also achieving a low standard deviation on the final rule-set size. The same does not hold for higher values of k as these datasets become increasingly more difficult when the attributes which compose their defining DNF rules increase in number. This increase in domain difficulty translates into an increase in the size of the rule-sets evolved by BioHEL's configurations, however the solutions sizes evolved by the LS settings still remain considerably lower than the ones generated by the Basic system. The increasing difficulty of the datasets can also be observed from the increase in the runtimes of the Basic configuration. For these datasets, the percentage of instances with class 0 also increases with k. And since the default rule of BioHEL always predicts class 0 for the kDNF datasets, for higher k values the search space consists of predominantly instances with class 0 with few isolated instances with class 1. This is why, for the continuous kDNF datasets with $k = \{8,9,10\}$ the average training accuracies start at very high values of over: 92%, 96% and 98% respectively. For these cases, there are no noticeable differences in final accuracies, but the plots reveal the extra search effort of the memetic configurations.

For $k = 5$ we can see that the best memetic configurations achieve better training accuracy than the Basic one while having similar runtime requirements. We can also observe that the Basic configuration has evolved a larger number of less general rules, which even exceeded the accuracy of the memetic configurations for a large portion of the runtime interval. However, towards the end of the learning process, the Basic system becomes stuck and is not able to evolve any new accurate rules while the memetic configurations continued exploration. Compared to the dataset variant with $k = 5$, for the one with $k = 6$ the differences between

Table 9 ES2: kDNF20-k{8,9,10}-r20 results

D.	Configuration	Train Accuracy	Runtime (s)	#Rules
kDNF20-k8-r20	Basic	100.00%±0.0%	832.91±110.4	21.00±0.0
	0.25-RC-RS	100.00%±0.0%	550.25±52.2	21.00±0.0
	0.25-RC-RS-RG	100.00%±0.0%	644.54±114.2	21.00±0.0
	0.25-RS	100.00%±0.0%	645.55±161.0	21.00±0.0
	0.25-RS-RG	100.00%±0.0%	649.33±72.7	21.00±0.0
	0.20-RC-RS-RG	100.00%±0.0%	727.60±230.6	21.00±0.0
kDNF20-k9-r20	Basic	100.00%±0.0%	966.87±160.8	21.00±0.0
	0.10-RC-RS	100.00%±0.0%	566.66±130.2	21.00±0.0
	0.15-RC-RS	100.00%±0.0%	569.76±64.9	21.00±0.0
	0.05-RC-RS-RG	100.00%±0.0%	575.41±57.1	21.00±0.0
	0.10-RC-RS-RG	100.00%±0.0%	589.05±66.7	21.00±0.0
	0.20-RS	100.00%±0.0%	615.68±94.6	21.00±0.0
kDNF20-k10-r20	Basic	100.00%±0.0%	1530.82±544.3	21.00±0.0
	0.15-RC-RS	100.00%±0.0%	943.37±265.7	21.00±0.0
	0.10-RC-RS	100.00%±0.0%	1042.72±259.6	21.00±0.0
	0.20-RC-RS	100.00%±0.0%	1046.89±319.5	21.00±0.0
	0.20-RC-RS-RG	100.00%±0.0%	1051.75±361.2	21.00±0.0
	0.15-RC-RS-RG	100.00%±0.0%	1080.84±325.8	21.00±0.0





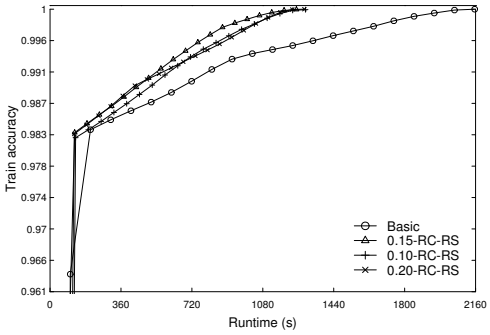
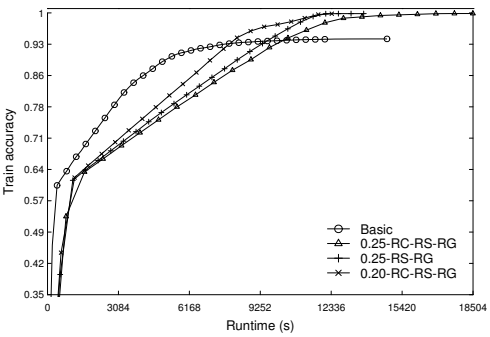
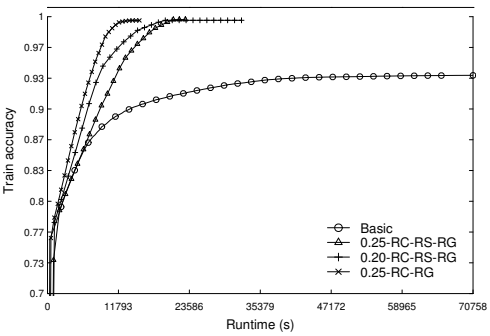
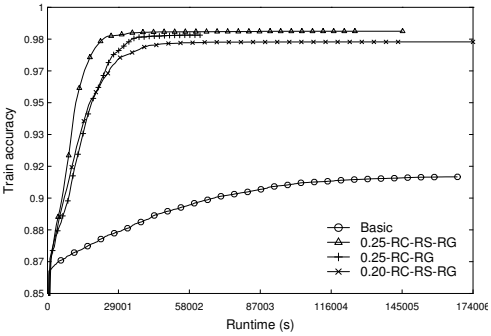


Table 10 ES2: ckDNF20-k{5,6,7}-r20 results

D.	Configuration	Train Accuracy	Runtime (10^3s)	#Rules
ckDNF20-k5-r20	Basic	94.00%±0.3%	10.71±1.8	63.00±7.1
	0.25-RC-RS-RG	99.86%±0.0%	12.21±2.7	21.30±0.7
	0.25-RS-RG	99.83%±0.0%	11.69±1.3	21.40±1.3
	0.20-RC-RS-RG	99.81%±0.0%	9.94±1.8	21.20±0.4
	0.20-RS-RG	99.80%±0.0%	10.47±2.9	21.20±0.4
	0.25-RC-RG	99.80%±0.0%	10.20±1.8	21.40±0.7
ckDNF20-k6-r20	Basic	93.64%±0.2%	47.46±13.0	169.70±15.2
	0.25-RC-RS-RG	99.69%±0.1%	17.90±4.8	22.10±1.4
	0.20-RC-RS-RG	99.61%±0.1%	16.01±7.6	23.80±2.8
	0.25-RC-RG	99.59%±0.1%	11.45±1.9	22.50±1.8
	0.25-RS-RG	99.58%±0.1%	15.81±6.3	23.50±2.8
	0.20-RS-RG	99.49%±0.1%	11.43±5.3	24.50±2.8
ckDNF20-k7-r20	Basic	91.11%±0.4%	120.27±33.6	459.20±52.8
	0.25-RC-RS-RG	98.75%±0.4%	78.96±43.8	113.10±54.8
	0.25-RC-RG	98.54%±0.2%	32.15±14.1	62.10±15.4
	0.20-RC-RS-RG	98.19%±0.5%	70.98±43.3	106.90±36.7
	0.20-RC-RG	97.67%±0.6%	41.35±19.9	87.30±16.4
	0.15-RC-RS-RG	97.59%±0.5%	57.81±16.2	126.40±23.9



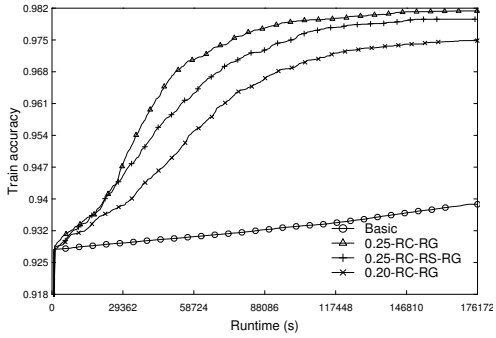


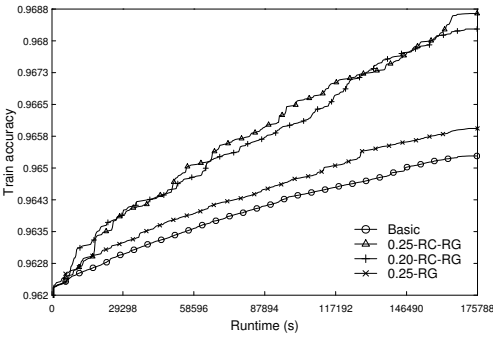


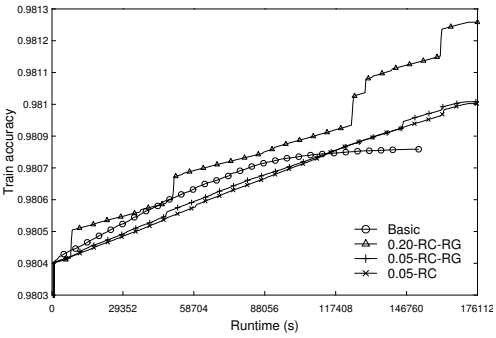
the Basic system and the best LS setting remain at approximately 6% for accuracy but increase for runtime complexity, with the best memetic configuration taking 2.65 times on average less seconds than the Basic system to converge to a final solution. The plot shows that the accuracy evolution of the Basic setting is inferior to the ones of the top five LS configurations - throughout the entire runtime interval. For this continuous kDNF dataset and for the one with a $k = 5$

Table 11 ES2: ckDNF20-k{8,9, 10}-r20 results

D.	Configuration	Train Accuracy	Runtime (10^3s)	#Rules
ckDNF20-k8-r20	Basic	93.84%±0.1%	170.48±6.4	499.50±63.6
	0.25-RC-RG	98.18%±0.5%	157.73±29.8	312.20±122.0
	0.25-RC-RS-RG	97.99%±0.3%	155.74±8.9	208.10±30.3
	0.20-RC-RG	97.51%±0.4%	157.63±21.0	326.10±81.8
	0.20-RC-RS-RG	97.44%±0.5%	169.06±2.3	242.80±78.2
	0.15-RC-RS-RG	96.98%±0.4%	166.90±7.5	331.70±57.9
ckDNF20-k9-r20	Basic	96.53%±0.1%	160.06±26.1	602.80±120.6
	0.25-RC-RG	96.87%±0.3%	169.87±3.6	360.00±68.8
	0.20-RC-RG	96.83%±0.2%	168.54±3.9	398.40±84.0
	0.25-RG	96.60%±0.1%	169.86±3.5	394.20±70.1
	0.20-RC-RS-RG	96.56%±0.1%	169.17±2.9	324.60±74.9
	0.15-RC-RG	96.56%±0.1%	151.65±18.5	288.50±29.0
ckDNF20-k10-r20	Basic	98.08%±0.0%	902.15±357.3	216.90±73.3
	0.20-RC-RG	98.13%±0.0%	1722.21±6.4	381.70±93.5
	0.05-RC-RG	98.10%±0.0%	1701.27±19.1	471.50±157.6
	0.05-RC	98.10%±0.0%	1710.88±28.1	508.00±92.4
	0.05-RC-RS	98.10%±0.0%	1690.84±55.7	562.00±106.2
	0.10-RC-RG	98.09%±0.0%	1641.17±83.9	310.60±54.6







the best memetic configurations achieved almost perfect accuracy on the train set. The best LS combinations for these two datasets consist of all three operators applied together, RC-RG and RC-RS, with relatively high application probabilities. For $k = 7$, a large discrepancy in all three metrics can be seen between the Basic system and the 0.25-RC-RG configuration which achieves almost full accuracy, in a quarter of the runtime of the Basic system while producing seven times less rules. From the plot we can see that the learning rates of the top LS configurations have been superior to the one of the Basic system. For $k = 8$ the curves of the learning rates of the memetic configurations ascend slower than in the previous datasets, which is expected as datasets increase in difficulty. However, the evolution of the Basic system's accuracy remains almost stagnant throughout the runtime interval, while the accuracies of the best memetic configurations monotonically increase. As observed in the previous datasets, the memetic configurations evolved more compact rule sets with higher accuracy, but in this case with similar runtime requirements as the Basic configuration.

In the final two continuous kDNF datasets, with $k = \{9, 10\}$ the differences in accuracy become insignificant, this being due to the very small portion of the problem space being occupied instances with class 1, where the coverage of each rule contributes very little to the overall training accuracy of the system. However, from the plots, we can observe that there was a noticeable difference in learning rate for the Basic and memetic configurations; and as expected, the slope of the evolution of accuracy of the best memetic configuration exceeded the one of the Basic configuration. For $k = 9$ the presence of the 0.25-RG in the top five configurations is due to the fact that it was able to stop learning quickly, but still achieve an accuracy comparable with the one obtained by the more equally balanced (in terms of specificity-generality ratio) configurations. In this case, the Basic system evolved more bloated solutions, while the memetic combinations generated rule sets of almost halved size but with better accuracy. For the last dataset with $k = 10$, the memetic configurations were able to learn more rules and run for more time than the Basic configuration. This is due to the Basic system's difficulty in evolving rules with very high specificity, caused by the MDL-based fitness function, which are needed to learn rules for this domain, thus ending the Basic system's runs sooner.

In the last four continuous kDNF datasets, with a k ranging from 7 to 10, the best memetic configurations slowly shift from the combination of all three operators to just using RC-RG (both with 25% individual-wise application probability). This suggests that as problems increase in difficulty, BioHEL has less need for added specificity pressure, so RS becomes superfluous. While this stage did not evaluate BioHEL's extensions' capabilities of avoiding overfitting, it showed that the best memetic configurations can produce more compact solutions and in less time than the Basic system.

Table 12 ES3: Performance metrics on the noisy variants of the ckDNF20-k5-r20 dataset. Best configurations are shown in bold. Abbreviations used: N. for noise type, (nt) for the noise-tolerant version of the operators and (std) for their standard version.

N.	Configuration	Test accuracy	#Rules	Runtime (10^2 s)
5% Input	Basic	92.82%±0.4%	115.16±10.9	263.45±99.1
	0.05-RS-RG (nt)	95.28%±0.4%	115.70±14.6	285.29±84.7
	0.05-RS-RG (std)	95.16%±0.4%	122.26±14.9	249.94±59.7
	0.10-RC-RS-RG (nt)	96.02%±0.3%	63.48±6.1	**126.29±34.8**
	0.10-RC-RS-RG (std)	95.60%±0.3%	128.63±13.0	441.00±117.2
	0.25-RC-RS-RG (nt)	**96.71%±0.4%**	**59.17±7.4**	210.03±62.3
	0.25-RC-RS-RG (std)	95.23%±0.3%	129.61±12.1	569.57±143.1
15% Input	Basic	91.75%±0.4%	156.55±9.8	379.54±96.7
	0.05-RS-RG (nt)	93.69%±0.4%	169.59±13.1	604.36±126.0
	0.05-RS-RG (std)	93.56%±0.4%	178.47±14.2	348.18±91.0
	0.10-RC-RS-RG (nt)	93.52%±0.4%	127.22±9.1	313.64±89.0
	0.10-RC-RS-RG (std)	**93.90%±0.5%**	187.32±15.2	561.47±165.6
	0.25-RC-RS-RG (nt)	93.33%±0.4%	**102.33±7.9**	**305.78±95.1**
	0.25-RC-RS-RG (std)	93.17%±0.5%	182.89±16.1	954.43±317.2
25% Input	Basic	90.93%±0.3%	180.57±10.1	**322.29±79.0**
	0.05-RS-RG (nt)	91.90%±0.5%	173.67±11.3	356.97±89.3
	0.05-RS-RG (std)	92.32%±0.4%	199.90±15.7	405.48±116.4
	0.10-RC-RS-RG (nt)	91.51%±0.4%	153.15±8.2	348.05±118.1
	0.10-RC-RS-RG (std)	**92.65%±0.4%**	206.98±16.2	657.16±170.0
	0.25-RC-RS-RG (nt)	90.39%±0.4%	**119.22±7.4**	473.46±139.1
	0.25-RC-RS-RG (std)	91.94%±0.5%	199.03±15.9	887.34±251.5
5% Output	Basic	88.23%±0.4%	106.95±8.3	341.31±124.5
	0.05-RS-RG (nt)	90.44%±0.5%	86.70±10.4	383.43±34.0
	0.05-RS-RG (std)	90.99%±0.4%	130.83±10.8	463.94±33.4
	0.10-RC-RS-RG (nt)	91.65%±0.4%	48.90±4.6	**188.92±22.9**
	0.10-RC-RS-RG (std)	91.43%±0.3%	154.96±12.7	671.98±134.8
	0.25-RC-RS-RG (nt)	**92.75%±0.3%**	**43.48±4.4**	240.10±25.8
	0.25-RC-RS-RG (std)	91.02%±0.3%	224.86±14.7	1398.25±191.3
15% Output	Basic	78.04%±0.4%	96.69±7.4	388.15±80.5
	0.05-RS-RG (nt)	80.25%±0.4%	94.70±7.2	295.85±113.5
	0.05-RS-RG (std)	79.74%±0.5%	113.11±9.2	241.91±69.6
	0.10-RC-RS-RG (nt)	80.86%±0.4%	72.14±7.5	**171.94±51.7**
	0.10-RC-RS-RG (std)	80.04%±0.5%	125.92±8.5	414.05±111.2
	0.25-RC-RS-RG (nt)	**81.39%±0.4%**	**61.86±6.8**	309.61±88.0
	0.25-RC-RS-RG (std)	79.83%±0.7%	159.48±9.7	949.76±261.2
25% Output	Basic	68.00%±0.4%	**70.02±5.9**	242.45±61.4
	0.05-RS-RG (nt)	69.82%±0.4%	78.34±6.6	302.47±107.2
	0.05-RS-RG (std)	69.06%±0.4%	81.69±6.6	**238.18±66.2**
	0.10-RC-RS-RG (nt)	**69.97%±0.4%**	75.78±6.1	255.18±70.9
	0.10-RC-RS-RG (std)	69.20%±0.4%	86.76±7.0	469.33±77.7
	0.25-RC-RS-RG (nt)	69.65%±0.5%	75.11±7.1	489.97±84.6
	0.25-RC-RS-RG (std)	68.97%±0.5%	99.75±7.8	620.90±118.3
5% Both	Basic	87.53%±0.4%	132.83±8.6	252.93±64.2
	0.05-RS-RG (nt)	89.05%±0.5%	113.89±10.3	395.87±161.1
	0.05-RS-RG (std)	89.87%±0.4%	160.61±10.0	346.11±130.7
	0.10-RC-RS-RG (nt)	89.94%±0.4%	75.83±7.1	**175.85±60.1**
	0.10-RC-RS-RG (std)	90.14%±0.4%	185.56±12.3	573.45±176.2
	0.25-RC-RS-RG (nt)	**90.64%±0.4%**	**63.18±6.0**	201.55±59.9
	0.25-RC-RS-RG (std)	89.54%±0.6%	233.27±16.1	1104.31±293.1
15% Both	Basic	76.97%±0.4%	115.01±7.4	**276.77±122.8**
	0.05-RS-RG (nt)	78.67%±0.4%	117.70±8.0	284.20±71.3
	0.05-RS-RG (std)	78.32%±0.4%	130.22±8.3	324.69±78.3
	0.10-RC-RS-RG (nt)	**78.89%±0.4%**	100.19±7.0	299.39±101.4
	0.10-RC-RS-RG (std)	78.44%±0.4%	140.91±8.1	654.28±209.7
	0.25-RC-RS-RG (nt)	78.79%±0.4%	**87.46±7.2**	420.97±101.2
	0.25-RC-RS-RG (std)	78.11%±0.5%	161.01±7.8	1078.96±273.2
25% Both	Basic	67.10%±0.3%	**82.60±5.7**	**282.10±73.9**
	0.05-RS-RG (nt)	68.31%±0.4%	91.75±6.1	382.13±98.3
	0.05-RS-RG (std)	67.80%±0.4%	92.93±6.0	282.91±77.3
	0.10-RC-RS-RG (nt)	**68.45%±0.4%**	90.73±7.4	333.97±92.2
	0.10-RC-RS-RG (std)	67.88%±0.4%	96.48±6.6	478.83±111.9
	0.25-RC-RS-RG (nt)	67.83%±0.5%	86.97±6.2	488.80±128.1
	0.25-RC-RS-RG (std)	67.50%±0.4%	104.91±7.1	623.23±138.9

6.3 Results for Experiments Stage 3: Noise-tolerant operators

This evaluation stage sought to determine if the noise-tolerant operators improve BioHEL's performance when compared to using the standard LS operators. Next, we present our findings in terms of accuracy, rule-set size and runtime over the nine different types of noise embedded into the ckDNF20-k5-r20 problem. The evaluated configurations are the noise-tolerant and standard variants of the best overall configurations of ES2 and ES1. The results of this stage are shown in table 12. Moreover, the statistical analysis of this result is reported in table 13.

Table 13 Statistical analysis of the results of ES3. Best configurations for each metric are reported in bold. Configurations with performance significantly worse than the best method are underlined (Holm test 95% confidence)

Metric	Accuracy	#rules	Run-time
Friedman p-value	3.06e-05	7.79e-09	2.227e-06
Configuration	Average ranks		
Basic	6.88	3.00	2.66
0.05-RS-RG (nt)	3.55	3.55	4.00
0.05-RS-RG (std)	4.55	5.11	3.11
0.10-RC-RS-RG (nt)	**2.33**	2.22	**1.88**
0.10-RC-RS-RG (std)	2.77	6.22	5.66
0.25-RC-RS-RG (nt)	2.77	**1.22**	3.66
0.25-RC-RS-RG (std)	5.11	6.66	7.00

As expected, test accuracy decreases when the probability of adding noise increases. This effect is more pronounced for the Output and Both types of noise, as randomly flipping the class of instances has a stronger distorting effect on the hyperspace of a problem than adding noise to attribute values. A consequence of the inherent difficulty of this dataset can be seen through the existing small range of differences between the Basic and the best memetic settings for all types of noise. Overall all noise types of noise, the majority of configurations having the best accuracy have been noise-tolerant memetic opereators combinations. Overall, the best accuracy has been obtained by the 0.10-RC-RS-RG noise-tolerant configuration, which is significantly better than the basic BioHEL and the standard 0.25-RC-RS-RG setting.

For the majority of types of noise (7/9) the most compact rule-sets are evolved by noise-tolerant LS operators, while in the other two cases, the Basic configuration stops its learning process early, producing solutions with even fewer rules. The best setting is 0.25-RC-RS-RG, which also had the second best accuracy rank. This setting manages to significantly outperform the three memetic settings on their standard configuration and also 0.05-RS-RG on its noise-tolerant version.

The noise-tolerant operators generally achieve much faster runtimes than their standard variants, which is most evident in the noise types with 5% and 15% probability. The dataset variants with 25% noise levels are difficult for BioHEL to learn, and this is reflected in the low testing accuracies and small solution sizes of all configurations especially for the Output and Both types of noise. Because of this, the differences in runtime between the different configurations are less prounounced in the datasets with 25% noise levels. Overall, the noise-tolerant 0.10-RC-RS-RG configuration has the lowest average rank, significantly outperforming 0.10-RC-RS-RG (std) and 0.25-RC-RS-RG (std).

This evaluation stage has shown that the noise-tolerant operators are very effective in terms of generalisation potential, as they generate compact rule set sizes with high accuracies, in noisy datasets. As the standard operator variants do not consider the existence of noise when editing rules, they should only be applied for completely noiseless domains, while the noise-tolerant variants are highly equipped for dealing with noisy data, as our results have shown. The statistical tests indicate that 0.10-RC-RS-RG (nt) is the overall best setting, as it was the best configuration for accuracy and run-time and the second best for solution size.

6.4 Results for Experiments Stage 4: Global comparison

This subsection presents the results of evaluating the best memetic configurations of the first and second experimental stages {0.05-RS-RG, 0.10-RC-RS-RG, 0.25-RC-RS-RG}, the basic BioHEL system and four alternative machine learning algorithms on a broad collection of both synthetic and real-world large-scale datasets. Ten-fold cross-validation has been employed in this stage of experiments, and we are reporting cross-validation accuracy in table 14 and average rule set/tree size in table 15. Random Forest is not included in the solution size comparison as all of its trees have the same size due to the functioning of the method.

The Friedman test rejected for both accuracy (with a p-value of 3.597e-12) and rule set size (with a p-value of 9.096e-16) that all methods had the same performance. The best method in terms of accuracy was the 0.10-RC-RS-RG memetic BioHEL configuration, while GAssist had the best rank for rule set size. The 0.10-RC-RS-RG configuration had a performance significantly better than all non-BioHEL methods, while GAssist generated rule sets significantly smaller than any other method.

The comparison between the memetic settings and the basic BioHEL should take into account both performance indicators together, as we can see that there is a trade-off between them: There is a clear relation between the load of memetic search (the probability of activating the memetic operators) and the rule set size. The more memetic search, the smaller the rule sets. However, this increase, especially in the real-world datasets, is followed by a decrease in accuracy, which especially affects the 0.25-RC-RS-RG con-

Table 14 ES4: Average test-set accuracy and standard deviation in %. Best configurations shown in bold. Underlined methods had a performance significantly worse than the best method at 95% confidence

Dataset	BioHEL				Other ML methods			
	Basic	0.05-RS-RG	0.10-RC-RS-RG	0.25-RC-RS-RG	GAssist	C4.5	RF	PART
mx20	**100.0±0.0**	**100.0±0.0**	**100.0±0.0**	**100.0±0.0**	**100.0±0.0**	**100.0±0.0**	**100.0±0.0**	**100.0±0.0**
ParMX	**100.0±0.0**	**100.0±0.0**	**100.0±0.0**	**100.0±0.0**	75.5±4.1	68.5±15.3	81.5±1.4	49.9±0.2
kDNF20_k5	**100.0±0.0**	**100.0±0.0**	**100.0±0.0**	**100.0±0.0**	**100.0±0.0**	99.8±0.0	99.9±0.0	**100.0±0.0**
kDNF20_k6	**100.0±0.0**	**100.0±0.0**	**100.0±0.0**	**100.0±0.0**	**100.0±0.1**	99.9±0.0	99.9±0.0	**100.0±0.0**
kDNF20_k7	**100.0±0.0**	**100.0±0.0**	**100.0±0.0**	**100.0±0.0**	99.2±0.5	99.8±0.0	99.9±0.0	**100.0±0.0**
kDNF20_k8	**100.0±0.0**	**100.0±0.0**	**100.0±0.0**	**100.0±0.0**	98.2±0.5	99.9±0.0	99.9±0.0	**100.0±0.0**
kDNF20_k9	**100.0±0.0**	**100.0±0.0**	**100.0±0.0**	**100.0±0.0**	97.8±0.4	99.9±0.0	99.9±0.0	**100.0±0.0**
kDNF20_k10	**100.0±0.0**	**100.0±0.0**	**100.0±0.0**	**100.0±0.0**	98.5±0.2	99.9±0.0	99.9±0.0	**100.0±0.0**
ckDNF20_k5	94.1±0.4	99.2±0.1	99.6±0.1	**99.8±0.0**	87.0±2.2	80.0±0.1	80.8±0.2	94.5±0.4
ckDNF20_k6	93.0±0.5	97.3±0.4	99.0±0.1	**99.6±0.1**	86.3±1.7	83.5±0.1	83.4±0.1	88.8±0.5
ckDNF20_k7	90.4±0.4	92.1±0.7	95.7±0.9	**98.3±0.3**	88.3±1.0	90.3±0.1	89.8±0.1	90.4±0.4
wav	**84.6±1.6**	83.8±1.6	83.0±1.6	83.5±0.8	80.7±1.6	75.0±2.3	82.0±1.8	78.0±2.0
pen	98.9±0.2	**99.0±0.3**	**99.0±0.3**	98.7±0.3	89.4±1.7	96.6±0.3	98.6±0.2	96.9±0.5
Adult	**86.0±0.4**	85.9±0.6	**86.0±0.6**	**86.0±0.5**	85.9±0.4	**86.0±0.4**	84.1±0.4	85.6±0.5
connect-4	77.8±0.5	77.9±0.2	78.0±0.4	77.6±0.4	78.0±0.4	**80.9±0.5**	79.9±0.4	79.2±0.5
SS1	**71.0±0.8**	**71.0±0.9**	70.6±0.9	69.8±0.9	66.3±1.1	55.1±0.7	63.8±0.8	58.9±0.7
CN-W1	75.7±0.3	**75.8±0.4**	**75.8±0.4**	75.7±0.4	74.8±0.4	68.6±0.4	74.1±0.5	70.9±0.6
CN-W2	**76.0±0.4**	**76.0±0.4**	**76.0±0.4**	75.9±0.4	74.7±0.5	68.4±0.5	74.4±0.5	69.5±0.3
CN-W3	76.3±0.5	**76.4±0.4**	76.3±0.5	76.2±0.4	74.6±0.5	68.1±0.4	74.7±0.5	69.9±0.3
CN-W4	**76.5±0.4**	**76.5±0.4**	76.4±0.4	76.3±0.3	74.8±0.4	68.2±0.3	74.5±0.4	70.0±0.5
kddcup	99.9±0.0	99.9±0.0	**100.0±0.0**	**100.0±0.0**	99.2±0.1	**100.0±0.0**	**100.0±0.0**	**100.0±0.0**
Ave rank	3.36	3.13	**2.84**	3.27	6.16	6.40	5.70	5.14

Table 15 ES4: Average rule-set size/number of tree leaves. Best configurations shown in bold. Underlined methods had a performance significantly worse than the best method at 95% confidence

Dataset	BioHEL				Other ML methods		
	Basic	0.05-RS-RG	0.10-RC-RS-RG	0.25-RC-RS-RG	GAssist	C4.5	PART
mx20	25.6±2.2	26.0±2.0	25.8±2.0	26.1±2.3	**17.0±0.0**	486.4±47.2	43.5±9.0
ParMX	382.0±15.9	333.4±15.9	320.6±16.1	297.4±15.1	80.4±18.5	4308.6±3920.0	**2.3±1.1**
kDNF20-k5	**21.0**±0.0	**21.0±0.0**	**21.0±0.0**	**21.0±0.0**	**21.0±0.0**	8145.3±69.0	124.1±11.6
kDNF20-k6	**21.0±0.0**	**21.0±0.0**	**21.0±0.0**	**21.0±0.0**	**21.0±0.1**	6566.6±19.7	232.1±15.0
kDNF20-k7	21.0±0.0	21.0±0.0	21.0±0.0	21.0±0.0	**20.6±2.6**	6071.1±156.8	334.7±17.2
kDNF20-k8	21.0±0.0	21.0±0.0	21.0±0.0	21.0±0.0	**17.9±4.1**	3501.6±122.5	304.7±14.2
kDNF20-k9	20.8±2.1	20.8±2.1	21.0±0.0	21.0±0.0	**10.8±2.9**	2751.0±47.4	343.6±21.7
kDNF20-k10	21.0±0.0	21.0±0.0	21.0±0.0	21.0±0.0	**6.0±1.2**	1785.4±58.7	289.9±11.2
ckDNF20-k5	66.8±8.9	35.9±6.1	23.6±2.1	**21.2±0.5**	31.4±11.6	42082.6±305.9	2964.1±117.2
ckDNF20-k6	160.9±26.9	57.0±12.0	35.4±7.5	22.9±2.3	**18.3±9.5**	33168.9±230.7	2841.2±84.3
ckDNF20-k7	276.0±220.2	334.7±95.2	164.1±60.4	118.9±62.2	**6.7±3.4**	18630.7±257.0	1892.8±73.2
Adult	33.1±2.5	33.8±2.2	35.0±2.3	35.0±2.4	**9.1±1.9**	622.9±79.7	1031.3±33.6
connect-4	48.0±4.1	49.1±3.6	48.8±3.5	46.6±5.5	**36.4±4.1**	4075.8±145.8	3683.1±45.8
pen	267.1±16.2	238.6±16.4	244.8±17.8	223.1±17.5	**11.3±5.1**	188.2±4.0	81.1±4.8
SS1	108.2±3.9	104.8±3.9	102.4±4.5	93.6±4.0	**33.0±4.9**	9142.3±98.6	3589.4±64.5
wav	219.5±15.8	252.2±14.3	257.5±19.0	247.0±19.8	**6.4±0.9**	290.7±9.3	88.4±7.1
CN-W1	115.5±3.4	110.0±6.1	105.7±7.0	92.9±8.1	**40.2±7.0**	22662.1±154.8	6271.9±271.4
CN-W2	116.6±3.6	110.9±6.4	106.0±7.7	94.5±7.7	**36.5±8.2**	22144.3±155.2	7889.5±263.4
CN-W3	117.3±3.7	113.0±4.9	106.9±6.6	95.2±8.7	**35.4±9.4**	21342.7±123.2	7450.8±51.0
CN-W4	118.6±3.1	113.6±6.0	107.6±7.1	92.6±8.3	**36.5±9.0**	20671.1±122.4	7006.0±47.1
kddcup	94.8±7.8	90.4±7.2	98.5±6.9	99.0±7.6	**9.7±3.9**	713.9±106.5	88.0±7.0
Ave rank	4.05	3.90	3.64	3.07	**1.33**	6.76	5.24

figuration. This setting would obtain the best accuracy rank when considering the synthetic datasets only, but becomes the worst of the three memetic settings when considering all datasets overall. The cause of this decrease in test accuracy is not due to overlearning, because this memetic setting presents lower training accuracy as well (not reported). The noise-tolerant memetic operators prefer to sacrifice a bit of training accuracy in order to create much more general rules, as it can be seen in the rule-set size results.

GAssist generates the smallest rule sets, but it never obtains accuracy levels that are better than any of the BioHEL variants. This was expected in both fronts. GAssist was designed to generate very compact rule sets, including in its fitness function a very strong generalisation pressure. However, in large-scale datasets that require very complex solutions it is expected to struggle because of its nature as a Pittsburgh system (trying to evolve a complete rule set at once), as it creates very long individuals which are very difficult to evolve. Finally, the other three machine learning algorithms (C4.5, RandomForest and PART) present a peformance worse than any BioHEL variant on most datasets, only managing to outperform them in very few datasets, all of them being among the smaller of the real-world datasets. Both C4.5 and PART generate solutions that are much larger than any of the evolutionary rule learning methods in most problems.

It is also interesting to observe the performance on the CN datasets across methods. As described before, all datasets try to solve the same problem, but with varying degrees of precision (attributes) in the representation. CN-W1 has 60 attributes, CN-W2 has 100, CN-W3 has 140 and finally CN-W4 has 180 attributes. The peformance of the machine learning methods across the different versions of the dataset is hence a reflection of their scalability capacity. The BioHEL variants (all of them) are the only methods able to increase their accuracy from one version of the dataset to the next (in increasing number of attributes), showing that this method is not constrained in this domain by the number of attributes of the problem.

7 Conclusions and further work

This paper studied the applicability of integrating memetic extensions into the BioHEL evolutionary learning system, by adapting three of GAssist's discrete-attributes memetic operators to BioHEL and by defining and integrating six new continuous-data memetic operators tailored to the attribute list knowledge representation into BioHEL. The large scale evaluation of these operators revealed that they guide BioHEL into evolving more compact solutions, with improved test set accuracies and by using less runtime. On small continuous domains we have determined that the LS operators enable BioHEL with an increased capacity of avoiding irrelevant features of the domain. We have also showed that the noise-tolerant variants of the continuous memetic operators produce more compact rule-sets in less time compared to their standard variants, but with similar accuracies. Finally, when compared to the performance of other machine learning methods, Memetic BioHEL showed very competent performance.

In future work we would like to study the possibility of integrating an upper limit on the amount of computational resources to be used by the memetic operators in each GA iteration. This approach would possess the potential of maximising the generalisation potential of each memetic operator while reducing its runtime. Moreover, it would be interesting to optimise further the parameter that controls the trade-off between accuracy and generality in the noise-tolerant memetic operators, as the results have shown that in real-world problems there is still some room for improvement.

Acknowledgements

We acknowledge the support of the UK Engineering and Physical Sciences Research Council (EPSRC) under grant EP/H016597/1. We are grateful for the use of the University of Nottingham's High Performance Computing Facility.

References

1. Jaume Bacardit. *Pittsburgh Genetics-Based Machine Learning in the Data Mining era: Representations, generalization, and runtime*. PhD thesis, Ramon Llull University, Barcelona, Spain, 2004.
2. Jaume Bacardit, Edmund K. Burke, and Natalio Krasnogor. Improving the scalability of rule-based evolutionary learning. *Memetic Computing*, 1(1):55–67, 2009.
3. Jaume Bacardit, David E. Goldberg, Martin V. Butz, Xaveir Llorà, and Josep Maria Garrell. Speeding-up pittsburgh learning classifier systems: Modeling time and accuracy. In *Parallel Problem Solving from Nature - PPSN 2004*, pages 1021–1031. Springer-Verlag, LNCS 3242, 2004.
4. Jaume Bacardit and Natalio Krasnogor. Empirical evaluation of ensemble techniques for a pittsburgh learning classifier system. In *Learning Classifier Systems*, volume 4998 of *Lecture Notes in Computer Science*, pages 255–268. Springer Berlin Heidelberg, 2008.
5. Jaume Bacardit and Natalio Krasnogor. A mixed discrete-continuous attribute list representation for large scale classification domains. In *GECCO '09: Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation*, pages 1155–1162. ACM, 2009.
6. Jaume Bacardit and Natalio Krasnogor. Performance and efficiency of memetic pittsburgh learning classifier systems. *Evolutionary Computation Journal*, 17(3):307–342, 2009.
7. Jaume Bacardit, Pawel Widera, Alfonso Márquez-Chamorro, Federico Divina, Jesús S. Aguilar-Ruiz, and Natalio Krasnogor. Contact map prediction using a large-scale ensemble of rule sets and the fusion of multiple predicted structural features. *Bioinformatics*, 2012.

8. George W. Bassel, Enrico Glaab, Julietta Marquez, Michael J. Holdsworth, and Jaume Bacardit. Functional network construction in arabidopsis using rule-based machine learning on large-scale data sets. *THE PLANT CELL ONLINE*, 23(9):3101–3116, September 2011.
9. Martin V. Butz. *Rule-based evolutionary online learning systems: learning bounds, classification, and prediction*. PhD thesis, Champaign, IL, USA, 2004. AAI3153259.
10. Kenneth A. De Jong and William M. Spears. Learning concept classification rules using genetic algorithms. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 651–656. Morgan Kaufmann, 1991.
11. Janez Demšar. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research*, 7:1–30, 2006.
12. Marco Dorigo and Thomas Stützle. *And Colony Optimization*. The MIT Press, Cambridge, Massachusetts, 2004.
13. Alberto Fernández, Salvador García, Julian Luengo, Ester Bernadó-Mansilla, and Francisco Herrera. Genetics-based machine learning for rule induction: State of the art and taxonomy and comparative study. *IEEE Transactions on Evolutionary Computation*, 14(6):913–941, 2010.
14. María A. Franco, Natalio Krasnogor, and Jaume Bacardit. Speeding up the evaluation of evolutionary learning systems using gpgpus. In *Proceedings of the 12th annual conference on Genetic and evolutionary computation*, GECCO '10, pages 1039–1046, New York, NY, USA, 2010. ACM.
15. María A. Franco, Natalio Krasnogor, and Jaume Bacardit. Analysing biohel using challenging boolean functions. *Evolutionary Intelligence*, 5(2):87–102, June 2012.
16. Maria A. Franco, Natalio Krasnogor, and Jaume Bacardit. Post-processing operators for decision lists. In *Proceedings of the fourteenth international conference on Genetic and evolutionary computation conference*, GECCO '12, pages 847–854, New York, NY, USA, 2012. ACM.
17. Maria A. Franco, Natalio Krasnogor, and Jaume Bacardit. Post-processing operators for decision lists. In *Proceedings of the fourteenth international conference on Genetic and evolutionary computation conference - GECCO '12*, page 847, Philadelphia, Pennsylvania, USA, 2012.
18. A. Frank and A. Asuncion. UCI machine learning repository, 2010.
19. Salvador García and Francisco Herrera. An Extension on "Statistical Comparisons of Classifiers over Multiple Data Sets" for all Pairwise Comparisons. *Journal of Machine Learning Research*, 9:2677–2694, December 2008.
20. John J. Grefenstette. Lamarckian learning in multi-agent environments. In Rick Belew and Lashon Booker, editors, *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 303–310, San Mateo, CA, 1991. Morgan Kaufman.
21. Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. The WEKA data mining software: an update. *SIGKDD Explorations*, 11(1):10–18, 2009.
22. George Harik. Linkage learning via probabilistic modeling in the ecga. Technical Report 99010, Illinois Genetic Algorithms Lab, University of Illinois at Urbana-Champaign, 1999.
23. George Harik, Fernando G. Lobo, and David E. Goldberg. The compact genetic algorithm. *IEEE-EC*, 3(4):287, November 1999.
24. Michael J. Kearns and Umesh V. Vazirani. *An introduction to computational learning theory*. MIT Press, Cambridge, MA, USA, 1994.
25. James Kennedy and Russell Eberhart. Particle swarm optimization. In *Neural Networks, 1995. Proceedings., IEEE International Conference on*, volume 4, pages 1942 –1948 vol.4, nov/dec 1995.
26. John R. Koza. *Genetic Programming*. The MIT Press, Cambridge, Massachusetts, 1992.
27. Natalio Krasnogor and James Smith. A tutorial for competent memetic algorithms: model, taxonomy, and design issues. *IEEE Trans. Evolutionary Computation*, 9(5):474–488, 2005.
28. Pedro Larrañaga and Jose Antonio Lozano. *Estimation of Distribution Algorithms*. Kluwer Academic, 2002.
29. Xavier Llorà, Anusha Priya, and Rohit Bhargava. Observer-invariant histopathology using genetics-based machine learning. *Natural Computing*, 8:101–120, 2009. 10.1007/s11047-007-9056-6.
30. Xavier Llorà, Kumara Sastry, and David E. Goldberg. The compact classifier system: Scalability analysis and first results. In *Proceedings of the Congress on Evolutionary Computation 2005*, volume 1, pages 596–603. IEEE Press, 2005.
31. Xavier Llorà, Kumara Sastry, Cláudio F. Lima, Fernando G. Lobo, and David E. Goldberg. Linkage learning, rule representation, and the x-ary extended compact classifier system. In *Learning Classifier Systems, Revised Selected Papers of IWLCS 2006-2007*, pages 189–205. LNAI 4998, Springer-Verlag, 2008.
32. Martin Pelikan, David E. Goldberg, and Erick Cantú-Paz. BOA: The Bayesian optimization algorithm. In *Proceedings of the Genetic and Evolutionary Computation Conference GECCO-99*, volume I, pages 525–532. Morgan Kaufmann, 1999.
33. Gilles Venturini. SIA: A supervised inductive algorithm with genetic search for learning attributes based concepts. In P. B. Brazdil, editor, *Machine Learning: ECML-93 - Proc. of the European Conference on Machine Learning*, pages 280–296. Springer-Verlag, 1993.
34. Steward W. Wilson. Classifier fitness based on accuracy. *Evolutionary Computation*, 3(2):149–175, 1995.
35. David Wyatt and Larry Bull. A memetic learning classifier system for describing continuous-valued problem spaces. In *Recent Advances in Memetic Algorithms*, pages 355–396. Springer, 2004.

Appendix

Table 16 ES1: Full cross-validation accuracy results on the Checkerboard datasets

Method	Number of attributes																			
	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Basic	97.1±1.8	97.1±1.7	97.1±1.4	97.2±2.2	97.3±2.0	97.2±2.1	96.5±2.3	94.9±3.2	94.0±3.2	92.8±3.6	90.6±4.4	87.5±6.3	84.3±6.9	81.4±6.6	81.2±9.9	84.4±9.0	82.2±13.4	82.9±11.3	81.5±13.3
0.05-RC	97.3±1.8	97.2±1.6	97.5±1.5	97.5±2.3	97.5±1.8	97.5±1.6	97.2±2.2	95.8±2.3	94.9±2.8	94.6±3.3	93.6±3.1	90.8±5.0	89.6±5.1	87.7±6.4	86.8±8.4	87.4±9.2	85.6±10.1	84.5±11.5	83.2±12.1
0.05-RS	97.5±1.6	97.3±1.6	97.7±1.5	97.8±1.9	97.5±1.6	97.6±1.5	97.7±1.8	97.2±1.9	97.1±1.6	97.2±2.0	96.9±2.2	95.3±3.1	94.9±3.0	93.9±3.0	92.5±4.7	91.6±6.5	90.5±7.8	87.8±10.2	88.2±10.1
0.05-RG	97.2±1.7	97.1±1.5	97.4±1.4	97.6±1.9	97.6±1.9	97.5±1.7	97.4±1.8	95.7±2.8	94.8±2.9	94.3±3.1	92.4±3.7	88.2±5.8	87.9±6.5	82.8±8.6	85.0±9.2	85.2±8.5	84.6±11.8	85.1±10.0	82.2±13.2
0.05-RC+RS	97.3±1.7	97.3±1.6	97.6±1.4	97.6±1.9	97.4±1.9	97.8±1.5	97.7±1.5	96.8±2.0	96.8±1.9	96.7±2.0	95.8±2.2	94.4±3.3	94.0±3.9	92.7±3.8	91.7±5.5	90.2±8.3	88.7±10.1	86.7±13.0	85.6±12.9
0.05-RC+RG	97.4±1.6	97.4±1.7	97.4±1.5	97.5±1.9	97.4±2.0	97.7±1.6	97.4±2.2	96.1±2.5	95.1±3.0	95.1±3.5	94.2±3.0	91.6±4.6	90.7±5.2	89.3±5.6	89.2±7.6	89.6±7.9	86.8±10.7	85.5±10.7	84.4±12.9
0.05-RS+RG	97.2±1.8	97.6±1.5	97.6±1.5	97.9±1.8	97.7±1.8	97.7±1.6	98.1±1.5	97.4±1.7	97.2±1.5	97.4±1.7	96.7±2.1	95.5±3.0	95.2±3.1	94.6±3.6	93.6±4.8	92.4±7.7	90.8±7.0	88.7±10.2	85.9±11.0
0.05-RC+RS+RG	97.3±1.6	97.5±1.5	97.4±1.7	97.7±1.9	97.6±1.6	97.7±1.5	97.9±1.5	97.0±1.8	97.0±2.0	97.0±2.0	96.0±2.9	94.5±3.3	94.9±3.4	94.1±3.2	92.6±5.6	92.0±7.2	91.7±7.7	89.2±9.9	85.4±12.1
0.10-RC	97.3±1.6	97.4±1.5	97.4±1.6	97.6±2.0	97.2±1.9	97.6±1.7	97.3±2.1	95.9±2.4	95.0±2.9	94.5±3.0	93.2±3.1	92.6±3.5	92.0±3.8	88.9±5.5	88.1±6.8	87.8±8.5	86.0±10.2	84.8±12.2	82.4±12.9
0.10-RS	97.3±1.6	97.4±1.6	97.6±1.4	97.8±1.8	97.7±1.9	97.6±1.6	97.9±1.5	97.3±1.7	97.3±1.9	97.4±1.6	97.3±1.9	96.5±2.9	96.4±2.3	95.1±3.3	93.1±5.2	90.7±9.3	89.3±8.6	86.9±12.3	83.3±14.6
0.10-RG	97.2±1.6	97.1±1.8	97.4±1.5	97.7±1.9	97.4±2.0	97.5±1.7	97.7±1.9	95.9±2.7	95.3±2.8	94.7±3.0	92.9±3.3	89.6±5.9	88.7±6.6	85.0±7.3	85.8±8.9	87.2±8.1	87.5±9.7	85.7±10.6	85.3±12.4
0.10-RC+RS	97.3±1.8	97.4±1.4	97.6±1.3	97.7±1.8	97.4±1.9	97.7±1.5	97.9±1.7	97.3±1.7	97.1±1.6	97.4±1.7	96.7±2.2	95.5±2.8	94.9±3.2	93.9±3.9	90.9±6.4	89.8±9.0	86.5±11.1	83.4±13.8	81.2±13.7
0.10-RC+RG	97.3±1.8	97.3±1.4	97.6±1.4	97.6±1.9	97.6±1.8	97.5±1.5	97.8±1.7	96.4±2.1	95.8±2.8	94.9±3.3	94.6±2.9	93.3±3.7	92.6±4.3	91.2±4.9	90.6±6.0	90.3±7.9	89.9±8.2	87.8±11.1	86.8±10.8
0.10-RS+RG	97.2±1.9	97.4±1.7	97.5±1.7	97.8±1.8	97.8±1.5	97.6±1.6	98.0±1.5	97.6±1.8	97.2±1.6	97.6±1.6	97.4±1.7	96.7±2.4	96.9±2.6	95.9±2.7	94.5±4.2	92.2±6.0	89.1±10.1	87.7±12.4	87.4±12.5
0.10-RC+RS+RG	97.4±1.7	97.2±1.6	97.5±1.6	97.6±2.0	97.4±1.9	97.8±1.6	98.0±1.5	97.3±1.7	97.1±2.0	97.1±2.0	97.0±2.0	96.2±2.7	95.9±2.9	95.2±3.1	93.2±5.4	90.8±8.1	87.4±10.2	84.3±15.0	85.7±12.2
0.15-RC	97.3±1.7	97.3±1.7	97.4±1.4	97.8±2.0	97.5±2.0	97.6±1.5	97.5±1.8	95.9±2.6	95.4±2.5	94.5±3.1	93.4±2.9	92.3±4.0	91.7±4.2	90.3±4.5	89.8±5.5	86.0±10.5	85.9±10.5	83.2±12.6	82.7±13.6
0.15-RS	97.3±1.9	97.4±1.7	97.8±1.6	97.7±1.9	97.7±1.8	97.6±1.4	98.1±1.3	97.5±1.7	97.2±1.8	97.6±1.7	97.2±1.8	96.8±2.4	96.6±2.9	95.4±3.3	91.3±8.6	87.3±12.0	85.5±12.0	81.4±15.8	80.0±15.9
0.15-RG	97.1±1.7	96.9±1.7	97.6±1.5	97.4±1.8	97.5±1.7	97.4±1.8	97.2±2.1	96.3±2.6	95.6±2.7	95.1±3.6	93.1±3.6	90.1±5.4	88.2±6.5	85.9±7.6	87.5±6.7	87.0±9.1	87.2±9.0	84.4±14.1	85.0±12.4
0.15-RC+RS	97.3±1.8	97.4±1.5	97.6±1.5	97.7±1.8	97.6±1.8	97.6±1.5	98.0±1.7	97.4±1.8	97.1±1.7	97.2±2.2	97.2±1.6	96.3±2.5	95.5±3.4	94.2±4.1	91.5±7.2	86.2±10.2	85.1±13.0	80.8±15.0	78.7±16.5
0.15-RC+RG	97.1±1.8	97.5±1.6	97.7±1.4	97.7±1.9	97.5±1.9	97.8±1.5	97.6±1.9	96.1±2.6	96.0±2.5	95.8±2.8	94.7±3.0	93.6±3.6	93.5±3.6	91.8±4.2	91.5±5.5	89.8±7.5	89.5±8.8	87.0±11.5	84.7±12.6
0.15-RS+RG	97.2±1.8	97.5±1.6	97.6±1.6	97.9±1.7	97.7±1.8	97.8±1.4	98.1±1.6	97.3±1.6	97.3±1.8	97.4±2.0	97.8±1.5	96.4±2.8	97.2±2.1	95.9±3.0	92.7±6.3	90.4±8.1	85.8±13.5	88.6±11.4	85.4±12.4
0.15-RC+RS+RG	97.3±1.8	97.2±1.7	97.5±1.5	97.6±2.0	97.5±2.0	97.8±1.5	97.9±1.6	97.5±1.5	97.3±1.8	97.5±2.1	97.2±1.9	96.5±2.6	96.6±3.0	95.7±2.8	91.3±8.5	89.3±9.3	87.9±10.8	83.2±14.1	80.7±15.3
0.20-RC	97.2±1.7	97.3±1.5	97.6±1.5	97.7±2.0	97.6±1.8	97.9±1.5	97.5±1.9	95.8±2.9	95.5±2.7	95.3±2.9	94.4±3.0	93.0±3.6	92.2±4.1	90.9±4.6	89.9±6.1	86.2±10.3	84.9±10.8	84.6±12.6	82.3±12.2
0.20-RS	97.5±1.6	97.5±1.7	97.7±1.3	97.9±1.8	97.8±1.9	97.6±1.6	98.0±1.6	97.6±1.7	97.1±1.9	97.5±1.7	97.4±1.7	96.4±2.8	96.3±2.9	95.1±4.1	90.3±7.7	85.3±12.4	84.5±15.4	83.4±13.1	78.3±14.5
0.20-RG	97.1±1.7	97.3±1.7	97.5±1.3	97.7±2.1	97.3±2.0	97.8±1.7	97.7±1.8	96.5±2.5	95.5±2.8	94.5±3.5	93.8±3.2	90.5±5.0	88.2±7.2	87.7±6.5	87.9±6.4	88.8±7.2	87.5±9.2	86.9±10.0	86.2±12.0
0.20-RC+RS	97.4±1.8	97.2±1.6	97.6±1.4	97.8±2.0	97.6±1.9	97.7±1.3	97.9±1.5	97.3±1.8	97.2±2.1	97.0±2.8	96.5±2.6	95.8±2.9	96.0±3.3	94.3±4.3	87.3±11.5	83.9±13.3	79.6±16.0	74.5±16.9	74.6±17.0
0.20-RC+RG	97.3±1.8	97.6±1.5	97.5±1.5	97.8±1.9	97.4±2.0	97.7±1.5	97.8±1.7	96.3±2.5	95.8±2.6	95.8±2.8	94.7±2.8	93.3±3.8	93.6±3.6	92.5±3.7	92.4±6.1	91.0±8.2	90.5±6.6	86.0±11.7	87.1±9.4
0.20-RS+RG	97.2±1.8	97.5±1.7	97.6±1.6	97.8±1.7	97.8±1.8	97.9±1.5	98.1±1.7	97.2±2.0	97.4±1.7	97.5±1.9	97.5±1.4	96.7±2.5	97.3±2.4	95.9±2.9	91.6±7.9	88.8±11.5	86.8±12.4	83.3±15.0	85.8±12.4
0.20-RC+RS+RG	97.2±1.9	97.5±1.4	97.7±1.3	97.8±1.9	97.7±1.9	97.7±1.3	98.1±1.3	97.5±1.7	97.4±1.7	97.4±1.8	97.2±1.8	96.7±2.5	96.7±2.7	95.4±3.6	91.1±8.9	85.1±12.3	85.8±13.2	84.6±14.0	77.3±17.6
0.25-RC	97.1±1.7	97.4±1.4	97.7±1.4	97.7±2.0	97.6±1.8	97.7±1.7	97.5±2.1	95.7±2.6	95.7±2.4	95.4±2.7	94.5±2.8	92.6±4.0	92.7±3.9	90.7±4.6	89.9±5.4	87.2±9.2	88.0±8.9	84.2±13.6	82.5±13.7
0.25-RS	97.4±1.7	97.6±1.6	97.7±1.5	97.9±1.8	97.7±1.7	97.7±1.5	98.0±1.5	97.5±1.7	97.2±1.8	97.2±2.8	96.6±2.8	96.2±3.1	95.8±3.8	95.3±3.8	83.2±12.8	84.4±12.9	81.3±15.4	77.9±16.8	75.3±16.6
0.25-RG	97.2±1.9	97.3±1.6	97.6±1.4	97.7±1.9	97.5±1.9	97.5±1.6	97.7±1.9	96.8±2.5	96.0±2.5	95.8±2.8	93.3±3.6	90.9±6.2	89.1±6.1	87.4±7.0	88.6±7.4	90.1±7.4	88.6±9.1	88.2±9.4	84.3±11.2
0.25-RC+RS	97.2±1.9	97.3±2.0	97.7±1.5	97.8±1.6	97.6±1.8	97.7±1.5	98.1±1.6	97.1±2.0	97.1±1.8	96.9±2.4	96.4±2.7	94.6±4.1	93.0±6.1	92.5±6.0	83.1±13.3	79.1±15.2	78.9±15.7	72.9±18.8	68.9±18.5
0.25-RC+RG	97.3±1.8	97.4±1.6	97.5±1.2	97.7±2.0	97.6±1.7	97.8±1.5	97.7±2.0	96.5±2.6	96.2±2.2	95.9±2.9	95.2±2.7	93.7±3.5	93.9±3.4	92.9±3.9	92.2±4.7	90.5±6.5	90.7±8.6	88.1±9.5	86.0±11.5
0.25-RS+RG	97.2±1.7	97.6±1.5	97.8±1.5	97.8±1.9	97.9±1.8	97.9±1.3	98.1±1.4	97.5±1.8	97.5±1.6	97.6±2.0	97.2±2.4	96.4±2.7	97.1±2.6	95.1±4.0	90.7±7.8	86.6±11.9	86.8±12.9	83.8±14.3	81.7±13.2
0.25-RC+RS+RG	97.1±1.7	97.4±1.7	97.5±1.5	97.7±1.9	97.8±1.7	97.8±1.5	98.0±1.5	97.5±1.6	97.3±1.9	97.3±1.8	96.6±2.5	95.9±3.2	95.9±3.5	94.6±4.5	88.9±10.0	85.2±11.5	81.9±15.2	79.4±15.1	76.8±17.4

Table 17 ES1: Full run-time (in s) results on the Checkerboard datasets

Method	Number of attributes																			
	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Basic	1.4±0.2	1.6±0.2	1.8±0.3	1.7±0.2	1.7±0.3	1.9±0.4	2.1±0.3	2.3±0.5	2.7±0.5	3.1±0.6	3.5±0.7	4.0±0.9	4.6±0.9	5.2±0.9	5.0±1.0	4.1±0.9	4.0±1.0	3.9±0.9	3.8±0.9
0.05-RC	1.5±0.2	1.8±0.3	1.9±0.4	1.9±0.4	2.0±0.3	2.0±0.3	2.2±0.4	2.5±0.5	2.8±0.5	3.2±0.6	3.4±0.6	3.8±0.7	4.4±0.7	4.9±1.1	4.4±1.0	4.1±1.0	3.9±0.9	4.0±0.9	4.0±1.0
0.05-RS	1.5±0.1	1.6±0.2	1.7±0.2	1.8±0.3	1.8±0.2	1.9±0.3	1.9±0.3	2.1±0.3	2.2±0.4	2.4±0.4	2.6±0.4	2.9±0.4	3.2±0.5	3.5±0.5	3.4±0.6	3.3±0.7	3.4±0.7	3.5±0.9	3.2±0.9
0.05-RG	1.7±0.2	1.9±0.2	2.0±0.3	2.0±0.3	2.0±0.3	2.1±0.3	2.3±0.3	2.6±0.5	2.9±0.6	3.3±0.6	3.6±0.7	4.6±1.1	5.0±1.2	6.1±1.6	5.2±1.2	4.9±1.2	4.7±1.4	4.4±1.1	4.4±1.3
0.05-RC+RS	1.6±0.2	1.8±0.3	1.9±0.3	2.0±0.4	1.9±0.2	2.1±0.3	2.1±0.3	2.2±0.3	2.4±0.4	2.8±0.4	2.9±0.4	3.3±0.4	3.6±0.6	3.9±0.6	3.8±0.7	3.7±0.8	3.6±0.7	3.8±1.2	3.7±1.1
0.05-RC+RG	1.6±0.2	1.9±0.3	2.0±0.4	2.0±0.4	1.9±0.3	2.1±0.3	2.2±0.3	2.5±0.4	2.7±0.5	3.0±0.5	3.2±0.5	3.8±0.8	4.2±1.0	4.8±1.0	4.3±0.9	4.1±1.0	4.0±1.1	4.2±1.3	3.9±1.1
0.05-RS+RG	1.7±0.2	1.8±0.2	1.9±0.3	1.9±0.2	1.9±0.2	2.0±0.2	2.1±0.3	2.2±0.3	2.3±0.4	2.4±0.4	2.7±0.5	3.0±0.5	3.4±0.6	3.6±0.6	3.5±0.7	3.5±0.8	3.5±0.8	3.6±0.9	3.7±1.1
0.05-RC+RS+RG	1.7±0.2	1.9±0.3	2.0±0.3	2.0±0.3	2.0±0.3	2.0±0.3	2.1±0.3	2.3±0.3	2.4±0.3	2.6±0.4	2.8±0.5	3.2±0.6	3.5±0.6	3.8±0.6	3.7±0.6	3.6±0.8	3.5±0.8	3.7±1.0	3.9±1.2
0.10-RC	1.6±0.2	1.8±0.3	1.9±0.3	2.1±0.4	2.1±0.4	2.2±0.5	2.4±0.4	2.6±0.5	3.0±0.6	3.3±0.6	3.6±0.5	4.1±0.7	4.4±0.6	5.2±0.9	4.7±0.9	4.3±0.9	4.1±1.0	4.4±1.3	4.2±1.2
0.10-RS	1.7±0.2	1.8±0.2	1.9±0.2	2.0±0.2	2.0±0.2	2.1±0.2	2.2±0.3	2.3±0.4	2.4±0.3	2.5±0.4	2.7±0.4	3.2±0.4	3.3±0.5	3.8±0.6	3.6±0.7	3.8±1.0	3.6±0.8	3.9±1.2	4.0±1.3
0.10-RG	1.9±0.2	2.2±0.3	2.3±0.3	2.2±0.3	2.3±0.3	2.4±0.3	2.7±0.4	3.0±0.5	3.3±0.5	3.7±0.6	4.2±0.8	5.4±1.4	6.0±1.4	7.1±1.7	6.1±1.7	5.4±1.6	5.1±1.3	5.0±1.5	4.9±1.5
0.10-RC+RS	1.7±0.2	1.9±0.2	2.0±0.2	2.1±0.3	2.2±0.3	2.3±0.3	2.4±0.3	2.6±0.3	2.8±0.4	3.1±0.4	3.4±0.4	3.8±0.6	4.2±0.8	4.2±0.8	4.2±0.8	4.1±0.9	4.0±1.0	4.5±1.5	4.5±1.4
0.10-RC+RG	1.9±0.2	2.1±0.3	2.2±0.4	2.3±0.3	2.3±0.3	2.4±0.3	2.7±0.4	3.0±0.5	3.1±0.5	3.5±0.5	3.7±0.7	4.1±0.7	4.5±0.8	5.2±1.3	4.7±1.1	4.5±1.1	4.1±1.0	4.5±1.3	4.3±1.3
0.10-RS+RG	2.0±0.1	2.2±0.2	2.3±0.2	2.3±0.2	2.4±0.2	2.4±0.2	2.5±0.3	2.7±0.3	2.8±0.3	2.8±0.3	3.0±0.4	3.5±0.5	3.8±0.6	4.1±0.7	3.9±0.7	4.2±0.8	4.2±1.2	4.6±1.5	4.3±1.2
0.10-RC+RS+RG	2.0±0.2	2.2±0.2	2.3±0.3	2.4±0.3	2.4±0.3	2.4±0.3	2.5±0.3	2.6±0.3	2.7±0.3	3.0±0.4	3.1±0.4	3.6±0.6	3.9±0.7	4.3±0.8	4.2±0.8	4.4±1.1	4.5±1.3	5.0±1.7	4.5±1.3
0.15-RC	1.6±0.2	1.9±0.4	2.1±0.4	2.1±0.3	2.1±0.3	2.3±0.4	2.7±0.5	2.8±0.4	3.2±0.5	3.6±0.7	3.8±0.7	4.4±0.8	4.9±0.8	5.3±0.9	4.9±0.9	4.9±1.2	4.4±1.0	4.8±1.4	4.4±1.3
0.15-RS	1.9±0.2	2.0±0.2	2.1±0.2	2.2±0.2	2.3±0.3	2.3±0.2	2.4±0.3	2.6±0.4	2.7±0.3	2.8±0.4	3.0±0.4	3.2±0.5	3.7±0.6	4.0±0.6	4.2±1.0	4.3±1.3	4.3±1.2	4.8±1.6	4.7±1.6
0.15-RG	2.2±0.2	2.6±0.4	2.7±0.4	2.6±0.3	2.7±0.3	2.9±0.4	3.2±0.5	3.5±0.5	3.9±0.7	4.2±0.8	4.8±1.0	6.0±1.5	6.8±1.7	8.0±2.0	6.9±1.8	6.6±1.9	6.0±1.7	6.2±2.1	5.7±1.8
0.15-RC+RS	1.9±0.2	2.1±0.2	2.2±0.3	2.2±0.2	2.3±0.3	2.5±0.3	2.5±0.3	2.7±0.4	2.8±0.4	3.0±0.4	3.3±0.4	3.6±0.5	4.1±0.6	4.5±0.7	4.5±0.9	4.7±1.2	4.6±1.4	5.1±1.7	5.0±1.8
0.15-RC+RG	2.1±0.2	2.4±0.4	2.4±0.3	2.6±0.3	2.6±0.3	2.8±0.3	3.0±0.4	3.4±0.5	3.6±0.6	3.9±0.7	4.3±0.7	4.7±0.9	5.1±0.8	5.6±1.0	5.2±1.0	5.1±1.3	4.9±1.3	5.2±1.6	5.3±1.8
0.15-RS+RG	2.4±0.2	2.6±0.2	2.7±0.3	2.7±0.2	2.8±0.2	2.9±0.2	2.9±0.2	3.1±0.3	3.2±0.3	3.3±0.3	3.5±0.3	3.8±0.5	4.2±0.7	4.6±0.7	4.8±1.0	4.8±1.1	5.2±1.7	5.1±1.4	5.1±1.6
0.15-RC+RS+RG	2.3±0.2	2.5±0.3	2.6±0.3	2.7±0.3	2.7±0.3	2.8±0.2	2.9±0.3	3.1±0.4	3.3±0.4	3.3±0.3	3.5±0.5	4.2±0.6	4.3±0.8	4.6±0.8	5.0±1.3	5.2±1.4	5.0±1.5	5.7±1.9	5.9±2.1
0.20-RC	1.7±0.2	2.0±0.3	2.2±0.4	2.2±0.3	2.3±0.4	2.4±0.4	2.8±0.5	3.1±0.5	3.4±0.6	3.6±0.6	4.0±0.7	4.4±0.6	5.0±1.0	5.8±0.9	4.9±0.9	5.2±1.2	4.9±1.3	4.8±1.4	4.8±1.4
0.20-RS	2.1±0.2	2.2±0.2	2.3±0.2	2.4±0.3	2.4±0.3	2.6±0.3	2.6±0.3	2.8±0.3	3.0±0.4	3.1±0.4	3.3±0.4	3.7±0.6	4.0±0.5	4.4±0.7	4.6±1.0	4.9±1.4	4.7±1.5	5.0±1.4	5.1±1.5
0.20-RC+RS	2.5±0.3	2.8±0.4	3.0±0.4	3.0±0.4	3.1±0.3	3.3±0.4	3.5±0.5	3.9±0.5	4.4±0.8	5.1±1.0	5.4±1.0	6.7±1.5	7.9±2.2	8.7±2.2	7.9±2.0	7.0±2.0	6.6±1.8	6.8±1.9	6.3±2.0
0.20-RC+RG	2.0±0.2	2.2±0.3	2.4±0.3	2.5±0.3	2.5±0.3	2.6±0.3	2.7±0.3	2.9±0.4	3.1±0.4	3.2±0.4	3.5±0.5	3.9±0.6	4.4±0.7	4.9±0.8	5.2±1.5	5.3±1.5	5.5±1.8	6.1±1.8	5.9±1.9
0.20-RS+RG	2.4±0.2	2.7±0.3	2.9±0.3	2.9±0.3	3.0±0.3	3.1±0.4	3.4±0.4	3.8±0.6	4.1±0.6	4.4±0.6	4.7±0.7	5.1±0.7	5.7±0.9	6.2±1.1	5.7±1.2	5.7±1.1	5.4±1.1	5.9±1.8	5.5±1.5
0.20-RC+RS+RG	2.8±0.2	3.0±0.2	3.0±0.2	3.2±0.2	3.2±0.2	3.3±0.2	3.4±0.3	3.6±0.5	3.7±0.4	3.9±0.4	4.0±0.4	4.4±0.6	4.7±0.7	5.2±0.9	5.5±1.3	5.9±1.9	5.8±1.8	6.4±2.1	5.7±1.8
0.25-RC	2.7±0.2	2.9±0.3	3.0±0.3	3.1±0.3	3.1±0.3	3.2±0.3	3.3±0.3	3.4±0.3	3.6±0.3	3.8±0.5	4.0±0.6	4.4±0.6	4.8±0.8	5.3±1.0	5.8±1.7	6.4±2.1	6.0±2.0	6.0±2.0	6.9±2.6
0.25-RS	1.7±0.2	2.0±0.3	2.2±0.3	2.3±0.3	2.3±0.3	2.5±0.4	2.9±0.5	3.2±0.6	3.6±0.7	3.8±0.7	4.3±0.7	4.9±0.7	5.3±0.9	6.1±1.0	5.4±0.9	5.1±1.0	4.8±1.1	5.0±1.4	4.8±1.5
0.25-RG	2.2±0.2	2.4±0.2	2.5±0.2	2.6±0.3	2.6±0.2	2.8±0.3	2.9±0.3	3.0±0.3	3.3±0.5	3.5±0.5	3.6±0.5	4.3±0.8	4.3±0.8	4.8±0.8	5.7±1.6	5.5±1.7	5.7±1.9	5.9±1.8	6.0±2.0
0.25-RC+RS	2.7±0.3	3.2±0.4	3.4±0.4	3.3±0.4	3.4±0.4	3.6±0.4	3.9±0.5	4.3±0.6	4.8±0.8	5.4±0.9	6.1±1.0	7.4±1.9	8.4±2.1	9.7±2.5	8.4±2.6	7.5±2.0	7.4±2.1	7.3±1.9	7.4±2.3
0.25-RC+RG	2.2±0.2	2.4±0.2	2.5±0.3	2.6±0.3	2.7±0.3	2.8±0.3	2.9±0.3	3.1±0.4	3.3±0.4	3.6±0.5	3.8±0.5	4.4±0.8	5.0±0.9	5.5±1.0	6.0±1.7	6.2±1.9	6.2±2.0	6.6±2.1	6.7±2.1
0.25-RC+RG	2.7±0.2	3.1±0.3	3.2±0.3	3.3±0.3	3.3±0.3	3.4±0.2	3.9±0.5	4.1±0.6	4.6±0.7	5.0±0.7	5.4±0.8	5.8±0.9	6.3±1.1	6.9±1.4	6.2±1.0	6.3±1.5	5.8±1.3	6.4±1.7	6.4±2.1
0.25-RS+RG	3.2±0.2	3.3±0.2	3.4±0.2	3.6±0.3	3.6±0.3	3.7±0.3	3.9±0.3	4.0±0.3	4.2±0.4	4.3±0.4	4.6±0.4	5.1±0.7	5.3±0.8	6.1±1.0	6.1±1.3	6.7±1.9	6.6±2.1	6.9±2.2	6.9±2.1
0.25-RC+RS+RG	3.0±0.2	3.2±0.2	3.4±0.3	3.5±0.3	3.5±0.2	3.6±0.2	3.7±0.3	3.9±0.4	4.1±0.4	4.2±0.4	4.5±0.6	5.1±0.8	5.6±1.0	6.0±1.1	6.5±1.7	7.0±2.3	7.3±2.7	7.5±2.6	7.6±2.8

Table 18 ES2: Full training accuracy results on the large-scale synthetic datasets

Method	Dataset													
	ckDNF20_k5	ckDNF20_k6	ckDNF20_k7	ckDNF20_k8	ckDNF20_k9	ckDNF20_k10	kDNF20_k5	kDNF20_k6	kDNF20_k7	kDNF20_k8	kDNF20_k9	kDNF20_k10	mx20	ParMX
Basic	94.0±0.3	93.6±0.2	91.1±0.4	93.8±0.1	96.5±0.1	98.1±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0
0.05-RC	95.9±0.4	94.9±0.5	91.4±0.4	94.0±0.3	96.4±0.0	98.1±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0
0.05-RS	98.3±0.2	95.9±0.4	91.3±0.8	93.5±0.3	96.4±0.1	98.1±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0
0.05-RG	98.3±0.3	95.9±0.7	91.7±0.6	93.7±0.3	96.4±0.0	98.1±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0
0.05-RC+RS	98.2±0.2	96.3±0.3	92.8±0.6	94.3±0.5	96.4±0.0	98.1±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0
0.05-RC+RG	99.3±0.1	97.6±0.3	93.0±0.6	94.4±0.4	96.4±0.0	98.1±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0
0.05-RS+RG	99.4±0.1	97.6±0.3	92.1±0.5	93.8±0.4	96.4±0.0	98.1±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0
0.05-RC+RS+RG	99.5±0.1	98.2±0.4	94.1±0.6	94.3±0.3	96.4±0.1	98.1±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0
0.10-RC	96.4±0.3	95.7±0.4	91.6±0.4	93.9±0.2	96.5±0.1	98.1±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	99.9±0.1
0.10-RS	98.9±0.1	97.2±0.2	92.1±0.3	93.5±0.2	96.4±0.0	98.1±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0
0.10-RG	99.1±0.2	97.4±0.3	92.3±1.1	93.7±0.2	96.5±0.1	98.1±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0
0.10-RC+RS	98.9±0.1	97.3±0.4	93.6±0.6	94.5±0.5	96.4±0.1	98.1±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0
0.10-RC+RG	99.6±0.1	99.0±0.2	94.4±0.7	95.1±0.4	96.5±0.0	98.1±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0
0.10-RS+RG	99.6±0.1	98.8±0.2	94.8± 1.2	93.6±0.2	96.4±0.0	98.1±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0
0.10-RC+RS+RG	99.7±0.0	99.2±0.1	96.1±0.6	95.9±0.5	96.5±0.1	98.1±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0
0.15-RC	96.6±0.4	96.0±0.3	92.1±0.7	94.2±0.3	96.4±0.0	98.1±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0
0.15-RS	99.2±0.1	97.8±0.3	92.1± 1.2	93.5±0.2	96.3±0.0	98.1±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0
0.15-RG	99.4±0.1	97.9±0.4	92.7± 1.1	94.1±0.3	96.4±0.0	98.1±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	99.9±0.1
0.15-RC+RS	99.1±0.1	97.9±0.2	94.5±0.7	94.9±0.4	96.4±0.0	98.1±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0
0.15-RC+RG	99.7±0.0	99.3±0.1	97.2±0.7	97.0±0.8	96.6±0.1	98.1±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0
0.15-RS+RG	99.7±0.1	99.2±0.2	94.7± 1.1	94.2±0.6	96.3±0.0	98.1±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0
0.15-RC+RS+RG	99.8±0.0	99.4±0.1	97.6±0.5	97.0±0.4	96.5±0.1	98.1±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0
0.20-RC	97.0±0.3	96.4±0.5	92.5±0.7	94.6±0.2	96.5±0.1	98.1±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0
0.20-RS	99.3±0.1	98.4±0.2	92.5±0.8	93.5±0.2	96.4±0.1	98.1±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0
0.20-RG	99.5±0.1	98.4±0.2	94.5± 1.6	94.0±0.4	96.5±0.0	98.1±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0
0.20-RC+RS	99.2±0.1	98.1±0.3	95.1±0.9	95.0±0.6	96.5±0.1	98.1±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0
0.20-RC+RG	99.7±0.0	99.5±0.1	97.7±0.6	97.5±0.4	96.8±0.2	98.1±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0
0.20-RS+RG	99.8±0.0	99.5±0.1	96.4± 1.6	93.6±0.4	96.4±0.1	98.1±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0
0.20-RC+RS+RG	99.8±0.0	99.6±0.1	98.2±0.5	97.4±0.5	96.6±0.1	98.1±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0
0.25-RC	97.1±0.2	96.4±0.3	93.1±0.5	94.9±0.4	96.5±0.0	98.1±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0
0.25-RS	99.5±0.1	98.4±0.3	92.9± 1.3	93.4±0.3	96.3±0.0	98.1±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0
0.25-RG	99.7±0.1	98.9±0.2	95.7±0.8	94.0±0.4	96.6±0.1	98.1±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	99.9±0.2
0.25-RC+RS	99.3±0.1	98.5±0.2	94.8±0.9	94.8±0.5	96.4±0.0	98.1±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0
0.25-RC+RG	99.8±0.0	99.6±0.1	98.5±0.2	98.2±0.5	96.9±0.3	98.1±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0
0.25-RS+RG	99.8±0.0	99.6±0.1	96.4± 1.2	94.0±0.6	96.4±0.1	98.1±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0
0.25-RC+RS+RG	99.9±0.0	99.7±0.1	98.7±0.4	98.0±0.3	96.5±0.1	98.1±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0

Table 19 ES2: Full run-time (in s) results on the large-scale synthetic datasets

Method	Dataset														
	ckDNF20_k10	ckDNF20_k5	ckDNF20_k6	ckDNF20_k7	ckDNF20_k8	ckDNF20_k9	ckDNF20_k10	kDNF20_k5	kDNF20_k6	kDNF20_k7	kDNF20_k8	kDNF20_k9	mx20	ParMX
Basic	10708.2± 1808.9	47455.9±129552	120268.2±33645.7	170482.6±6415.4	160058.0±26064.4	902147.3±35729.3	692.3±232.8	840.1±127.1	868.7±190.5	832.9± 110.4	966.9± 160.8	1530.8±544.3	38.4±5.0	2102.9± 345.5
0.05-RC	7773.9±1730.3	25443.9±4367.0	140780.8±33300.8	152654.6±13656.1	167682.3±8586.6	171088.2±2808.9	733.2±231.6	679.4±318.3	943.9±303.0	1016.2±295.2	781.9± 147.1	1378.9±487.9	40.9±7.6	2182.1± 167.2
0.05-RS	10426.8±1665.7	17530.8±6289.4	134977.0±34940.0	168781.6±2179.1	164832.5±12737.3	157941.7±43264.4	797.8±252.7	787.8±278.4	1149.4±110.1	995.9±196.4	682.7± 157.7	1340.7±167.8	35.3±4.5	1353.5± 187.7
0.05-RG	7267.3±2235.5	20967.2±5405.6	132837.9±22756.9	164292.1±10717.3	387760.9±38095.5	79423.7±29003.6	742.4±253.1	709.5±323.3	1242.9±97.3	1007.1±215.5	812.9± 71.6	1940.4±283.7	46.6±4.9	2509.5± 641.0
0.05-RC+RS	11828.3±4333.9	21377.7±5959.9	109785.3±30663.0	169669.3±3713.8	155434.5±17361.4	169084.5±5567.7	705.1±240.7	849.1±241.4	1055.9±225.1	742.6± 209.7	636.1± 125.6	1135.1±273.0	36.5±4.5	1288.8± 86.2
0.05-RC+RG	8688.9±3048.5	13275.8±4241.7	100526.7±27745.9	160136.8±12877.9	156780.0±27742.1	170127.1±1909.8	678.0±301.6	816.6±264.0	1310.9± 17.2	1057.2±379.0	811.1± 130.6	1585.3±318.9	39.6±5.4	1987.8± 191.1
0.05-RS+RG	7416.4±1870.7	16622.1±4158.2	150030.3±14081.7	169089.7±6557.2	157227.9±25765.0	195624.1±51196.7	663.1±262.4	857.4±240.8	1252.6± 35.5	1094.5±319.1	716.6± 176.0	1365.7±379.6	30.5±3.8	1291.6± 62.2
0.05-RC+RS+RG	6910.2±1766.7	16628.8±3354.8	140927.8±15406.3	163109.1±9910.8	154016.1±17172.6	169814.1±1308.8	866.4±172.8	1035.8±188.4	1249.7±29.8	830.7± 166.6	575.4±57.1	1086.0±214.9	35.6±3.6	1333.9± 127.9
0.10-RC	6643.2±582.4	22883.7±3317.8	152955.7±15325.8	162879.9±7064.6	159198.2±7531.6	169631.8±2669.8	714.6±239.4	1165.2±49.0	964.7±287.8	1098.6±267.3	840.8± 211.6	1553.1±366.7	42.3±6.5	2201.2± 172.0
0.10-RS	10927.7±4347.2	18556.6±3179.2	157065.2±11838.1	164568.1±8314.0	162733.4±4245.9	166789.5±9569.5	749.9±249.3	1198.5±31.9	850.4±229.1	905.6± 222.4	623.8±51.7	1275.8±218.5	31.8±5.9	1183.1± 128.6
0.10-RG	6919.6±836.6	23434.1±7194.7	161469.1±4972.4	159470.9±9543.7	158594.8±7939.0	93624.4±48467.9	779.6±234.9	1242.2±60.9	1270.3±234.1	1407.0±353.8	997.7±213.6	1889.9±502.3	44.1±19.5	2376.0± 240.4
0.10-RC+RS	12851.1±1551.8	14048.6±5246.7	111158.8±30274.4	165878.1±6675.1	155967.8±5785.8	170477.1±1181.1	714.9±258.0	1152.2±61.1	1133.2±203.4	888.2± 247.9	566.7± 130.2	1042.7±259.6	33.8±3.8	1136.1± 123.6
0.10-RC+RG	7993.9±1628.1	8327.0±1924.7	101824.2±20704.7	163728.9±14927.7	154090.9±23898.2	164117.4±8390.2	767.9±256.9	1222.0±52.1	1209.0±235.2	1046.2±288.4	784.6± 136.8	1519.6±356.1	40.6±3.9	2088.5± 143.4
0.10-RS+RG	9373.6±1787.1	19565.0±2333.1	124809.9±33759.2	164876.0±8125.1	160938.0±5319.0	166800.2±9239.1	858.2±275.1	839.3±281.2	1098.6±319.7	808.3± 186.5	724.1± 127.9	1262.5±376.9	30.6±3.8	1176.8± 166.1
0.10-RC+RS+RG	9195.7±1495.9	13212.6±2111.5	97197.6±26554.3	159662.1±11615.4	153963.5±21958.6	171271.7±1430.7	726.8±269.4	1031.8±193.4	1207.0±242.0	842.0± 180.4	589.1±66.7	1124.8±236.8	29.9±3.9	1083.1± 79.3
0.15-RC	11216.8±2708.3	19928.9±5501.9	122654.3±32968.4	167714.1±7900.1	152882.3±21011.5	160342.0±17789.5	691.4±275.3	780.0±269.8	1215.1±209.6	901.4± 194.6	831.7± 200.4	1368.2±458.2	37.8±3.7	2284.5± 183.4
0.15-RS	11458.5±2605.9	15073.1±3272.5	144206.0±20289.1	160267.6±10288.1	160892.6±6350.5	168787.4±3328.4	833.6±243.8	1147.8±168.4	1141.8±241.9	952.4± 218.9	635.6±63.9	1157.3±299.9	32.9±4.6	993.8± 41.0
0.15-RG	11180.3±3120.0	23426.0±3620.2	151311.4±15006.6	165261.3±7963.9	164059.7±5965.7	111424.8±49874.6	784.4±237.9	1287.4±27.0	1620.7± 79.7	1241.5±422.9	962.0± 127.4	2299.3±207.4	45.0±7.6	2555.8± 395.1
0.15-RC+RS	11104.5±1885.7	15633.5±1511.4	111702.9±30524.0	153146.9±20596.6	155938.4±10401.5	181826.3±48524.6	896.7±256.5	1119.2±130.5	650.8± 179.0	1000.5±188.6	569.8±64.9	943.4±265.7	30.6±2.4	1123.2± 119.8
0.15-RC+RG	7490.0±1317.6	10628.1±3619.8	39538.3±7830.7	157841.4±18572.3	151649.9±18519.3	165564.2±8977.5	846.1±229.2	1133.6±210.2	1066.4±280.9	1125.7±220.1	890.8± 115.6	1453.1±248.8	38.8±4.9	2189.2± 219.2
0.15-RS+RG	8143.2±1319.3	15251.8±4261.8	103904.3±25196.6	165663.2±9214.4	161733.0±4276.3	180129.3±88061	692.4±227.7	1233.2±71.0	1040.9±364.2	904.3± 184.4	796.3± 145.4	1414.8±358.1	32.7±4.2	1043.8± 179.7
0.15-RC+RS+RG	7840.8±1169.8	14927.4±4977.4	57809.7±16240.9	166902.5±7459.7	155208.0±19018.3	169468.7±2820.5	874.1±225.2	1179.6±160.2	1255.0±117.4	826.4± 190.0	619.2± 152.0	1080.8±325.8	29.4±5.3	1001.5±57.3
0.20-RC	8566.9±1186.1	30284.5±6890.0	77812.4±25754.7	162768.0±9757.5	163006.4±25959.5	169333.9±5080.3	787.8±248.1	1044.2±237.8	1429.0± 51.9	1052.8±305.6	893.6± 162.9	1397.3±334.0	40.1±2.7	2079.8± 154.8
0.20-RS	11641.3±4831.7	22677.7±6839.4	134938.7±26657.1	161450.1±9762.1	146111.5±34591.5	163857.9±13427.5	952.5±215.8	955.6±328.4	1183.0±224.7	932.4± 276.6	615.7±94.6	1350.8±323.2	33.7±7.0	975.4±124.6
0.20-RG	11819.1±3655.6	24753.1±8848.4	138952.3±22271.2	166764.4±3251.0	171422.6±1946.9	121453.7±39499.5	904.6±253.8	1232.5±280.3	1675.4±137.3	1150.1±179.8	1150.1± 134.4	2334.4±191.0	42.3±3.3	2563.0± 248.0
0.20-RC+RS	9215.4±1915.5	14352.4±4047.8	75941.9±28023.9	160683.2±11489.8	166872.5±13357.6	165389.7±10912.9	742.6±266.4	1088.5±260.8	1084.0±131.0	990.2± 224.2	681.3± 107.1	1046.9±319.5	30.4±3.2	1027.1± 77.8
0.20-RC+RG	10551.2±2973.7	10190.1±2101.4	41349.3±19940.9	157626.1±21021.3	168541.7±3939.7	172220.7±643.7	866.3±299.0	985.9±329.5	1222.6±368.7	1141.9±322.3	930.9± 205.2	1409.3±401.7	40.3±3.6	2264.2± 214.9
0.20-RS+RG	10465.0±2861.8	11428.6±5276.8	111917.5±34498.8	159015.9±14361.1	165751.3±12966.1	164146.7±11800.2	865.7±270.0	1197.0±233.0	935.6±302.6	845.3± 202.7	799.1± 182.7	1658.2±74.9	29.5±2.2	978.5± 69.6
0.20-RC+RS+RG	9935.7±1834.0	16015.0±7644.3	70984.9±43292.2	169060.7±2278.0	169170.0±2921.0	163786.3±8895.5	893.8±270.1	1142.2±226.7	919.5±251.4	727.6± 230.6	694.3±95.8	1051.8±361.2	29.3±3.0	1072.7± 169.4
0.25-RC	9926.6±6180.8	22203.7±8620.2	79726.5±13583.0	162686.9±14018.3	168187.3±3925.1	170401.6±2101.6	822.8±243.7	1156.6±82.7	1121.6±328.6	948.4± 280.4	955.5± 171.7	1209.3±416.7	39.0±4.8	2090.1± 159.9
0.25-RS	9579.7±1375.2	22701.5±12778.6	95084.4±19471.8	165825.3±9712.8	165136.1±15436.6	170184.1±2986.3	692.1±238.8	1216.7±34.6	1159.1±216.4	645.5± 161.0	630.5±97.7	1324.0±303.0	30.5±2.8	943.1± 139.7
0.25-RG	11496.8±1618.4	14506.3±4808.8	115117.7±31634.8	161582.2±6360.2	169863.4±3497.8	117218.5±48363.0	725.7±244.9	1206.1±143.6	1541.7±567.3	1119.0±235.3	1165.3± 139.1	2507.5±409.6	40.6±5.3	2754.6± 167.0
0.25-RC+RS	11337.5±2169.5	14444.1±5352.2	60221.2±20876.8	165422.1±9456.3	169910.3±2329.9	161682.0±15729.6	829.0±180.5	1127.5±154.3	883.9±263.3	550.3± 52.2	620.7± 102.3	1103.8±221.8	29.6±3.0	928.5± 100.3
0.25-RC+RG	10200.2±1796.6	11446.5±1893.8	32146.2±14126.9	157733.4±29824.2	169867.3±3589.8	162402.7±15308.8	668.0±223.1	1142.1±254.5	1218.2±392.4	820.4± 139.3	865.9± 127.4	1475.2±438.6	37.8±3.7	2170.1± 307.3
0.25-RS+RG	11687.0±1333.5	15814.1±6335.6	106587.1±30994.2	166776.7±6137.2	165779.9±13044.6	157766.8±14328.8	1011.2±99.2	1035.4±255.3	1229.0±298.8	649.3± 72.7	776.0± 103.0	1480.2±360.3	31.4±3.4	1022.9± 141.1
0.25-RC+RS+RG	12206.8±2664.4	17903.5±4791.9	78961.4±43798.8	155739.0±8872.5	165641.1±13270.5	164916.1±8376.1	698.8±189.3	961.0±160.9	1181.5±267.9	644.5± 114.2	705.4± 161.1	1311.7±89.7	29.7±2.9	1119.2± 132.1