Automatic Tuning of Rule-based Evolutionary Machine Learning Via Problem Structure Identification

Maria A. Franco, Natalio Krasnogor and Jaume Bacardit
The Interdisciplinary Computing and Complex BioSystems (ICOS) research group
School of Computing, Newcastle University, UK

Abstract

The success of any machine learning technique depends on the correct setting of its parameters and, when it comes to large-scale datasets, hand-tuning these parameters becomes impractical. However, very large-datasets can be pre-processed in order to distil information that could help in appropriately setting various systems parameters. In turn, this makes sophisticated machine learning methods easier to use to end-users. Thus, by modelling the performance of machine learning algorithms as a function of the structure inherent in very large datasets one could, in principle, detect "hotspots" in the parameters' space and thus, auto-tune machine learning algorithms for better dataset-specific performance. In this work we present a parameter setting mechanism for a rule-based evolutionary machine learning system that is capable of finding the adequate parameter value for a wide variety of synthetic classification problems with binary attributes and with/without added noise. Moreover, in the final validation stage our automated mechanism is able to reduce the computational time of preliminary experiments up to 71% for a challenging real-world bioinformatics dataset.

1 Introduction

Many machine learning techniques are tied to a series of hyper-parameters and/or selection of sub-components that need to be tuned. This challenge can broadly be defined as the *algorithm configuration problem*. When handling large-scale datasets, finding the adequate set of hyper-parameter values becomes a very expensive experimental process. Therefore, automatic hyper-parameter setting approaches are necessary in order to avoid a time-consuming preliminary experimentation stage, reduce the number of hyper-parameters that need to be set, and make these techniques more accessible to end-users. The creation of automated methods for the hyper-parameter tuning of machine learning algorithms but also of metaheuristic optimisation algorithms has been a very active area of research in recent years [1, 2, 3, 4].

This paper focuses on the specific context of evolutionary machine learning (EML). EML algorithms have shown to be very competitive methods for machine learning [5, 6] and have been applied to a very broad variety of real-world problems [7, 8, 9, 10, 11, 12, 13]. Hyper-parameter setting is a widespread problem in this field, since these systems use a genetic algorithm (GA) and fitness functions that often involve many hyper-parameters. Besides the generic approaches for hyper-parameter tuning mentioned above, there is a variety of methods that are specific to evolutionary computation [14]. Several techniques have been used such as reinforcement learning [15], or self-adaptive approaches [16, 17, 18, 19, 20, 21], among others. Moreover, even

Corresponding Author: Jaume Bacardit (jaume.bacardit@newcastle.ac.uk)

though theory exists about EML systems [22, 23, 24] that explains how these systems should be parameterised, only recently these theoretical works have started to be applied to fully tune these algorithms using a few synthetic problems as test scenarios [25, 26].

This paper proposes a heuristic procedure for the application of theoretical models of EML to automatically estimate the structure of classification problems with binary attributes and, from such estimates, set hyper-parameters accordingly. This work focuses on a specific EML algorithm called BioHEL [27]. This system has shown competent performance in very complex real-world domains [28, 29, 7, 8, 9, 10]. One of the main characteristics of this system is its fitness function, which tries to balance the accuracy, complexity and coverage of the rules in the solution. The key element of this fitness function is the *coverage breakpoint* hyper-parameter, which determines how many instances in the dataset a rule should cover to be considered good enough. Previous studies have shown that learning can be facilitated when this hyper-parameter is set correctly, in contrast to an incorrect setting which can push the system towards overgeneralisation [30]. Moreover, this hyper-parameter is highly problem dependant and finding its adequate value requires an extensive preliminary experimentation.

However, often the data holds the key of how to parameterise the system correctly. For example, as it was shown by [30] it is possible to set the adequate coverage breakpoint for classification problems with binary attributes (for the sake of compactness we will refer to these as *binary problems* in the rest of the paper) in BioHEL if the structure of the problem is known. But is it possible to define a generalisation about the structure of binary problems? A useful framework to model binary problems is the use of k-Disjunctive Normal Form (k-DNF) formulas, which are binary formulas that have r disjunctive terms and each one of these terms has k relevant variables of attributes expressed from the d possible ones. Many boolean benchmarks can be expressed as k-DNF formulas, and even complex real-world problems. Moreover, based on this structure (k and r) it is possible to create models that explain the behaviour of the BioHEL system [31]. Nevertheless, determining these values (k and r) is not straightforward and calculating their exact value would involve applying data mining over the problem which is an extra computational cost that we wish to avoid.

In this work we introduce an automated heuristic approach to determine the structure of an unknown binary problem (k and r) at runtime. This heuristic combines observations from the data and from a sample of randomly initialised rules evaluated against this data, to retro-feed theoretical models of the behaviour of the system [30, 31] and classify the problems into groups called *kr-groups* with a particular k and r associated.

In this particular work we show how this methodology can help adapt the coverage breakpoint hyper-parameter of BioHEL. Our experiments show how this mechanism can characterise challenging binary problems with and without noise. Moreover, we show how this mechanism can help adapt hyper-parameters of the system to solve a real-world protein structure prediction problem and reduce the total experimental time up to 71%, hence saving a lot of computational time and human effort.

2 Related Work

The automatic configuration and tuning is a very challenging process in evolutionary algorithms (EAs), where it has been studied in great length. In this section we first describe, from a general EA perspective, the different approaches to hyper-parameter control (and some examples of each). Afterwards, we focus on specific examples of hyper-parameter control in EML systems, which is the particular context of BioHEL. In terms of nomenclature we use a variety of terms (tuning, adjustment, control, configuration) because any one of these individual terms does not capture all the relevant literature in this area.

There are different types of hyper-parameter control algorithms. [14] presented a classifi-

cation for hyper-parameter control where the different techniques can be classified depending on what is changing (representation, mutation or crossover rates, selection mechanisms, etc.), or depending on how the change is made (deterministic, adaptive or self-adaptive). The *deterministic* techniques are the ones that adjust hyper-parameter without any feedback from the algorithm being controlled. The *adaptive* techniques are the ones that use some sort of feedback from the search process (i.e. during the optimisation/learning process). The *self-adaptive* techniques are the ones which evolve the hyper-parameters along with the rest of features of the problem. Using this classification our approach can be classified as a deterministic one, the methods described below applying the 1/5 rule for mutation rate adjustment [32, 33] would be adaptive, and [34] would be an example of the self-adaptive approach.

In the area of EAs, the earliest automatic hyper-parameter setting approaches were presented by [32] and [33], in which the mutation rate was adapted according to the *1/5th rule*. If the rate of successful mutation was over 1/5 the mutation rate was increased and if it was below this value it was decreased. Other early approaches involve the adaptation of crossover rate depending on how good were the resulting offsprings [35]. Regarding operator selection within evolutionary algorithms, [34] presented a very simple self-adaptive crossover-selection method. One extra bit in the classifier encoding represented the crossover that should be applied. Other approaches [36, 37] use rules to modify the hyper-parameters of local search operators (crossover and mutation).

Self-adaptive approaches have also been used in memetic algorithms to adapt the local search operators depending on the stage on which the search process is [18, 19, 20, 21].

Reinforcement learning (RL) has also been used to adapt hyper-parameters in EAs to identify the appropriate step size for the *1/5th rule* when adapting the mutation rate [38]. Furthermore, other more complex approaches [39] use RL to adapt the hyper-parameters of the GA considering not only the quality of the solutions, but also the cost incurred by the selected search operators.

There are several examples of self-adaptive mechanisms in the EML context. The Zeroth Level Classifier System (ZCS) [40] was extended with self-adaptive mutation in [41], to give an independent mutation rate to each classifier. Afterwards, this work was extended by self-adapting all the hyper-parameters in ZCS (mutation, learning rate, tax rate and discount factor) at the same time [42]. While in stationary environments the results are as good as the ones obtained with fixed hyper-parameters, in the more dynamical ones the self-adaptation improves the performance of the system. Self-adaptive mechanisms for both the mutation and the learning rate in XCS [43] were investigated in [44]. The self-adaptive mutation rate solved some generalisation problems when XCS was applied to long rule chain environments. Nevertheless, the performance was still sub-optimal in this case. Also, the system showed worse performance when trying to adapt the learning rate. Finally, Self-adaptive mutation has also been applied to the XCSF [45] using hyper-ellipsoidal condition structures [46], in which each part of the knowledge representation had its own self-adapted mutation rate.

More generally, the automated tuning of machine learning algorithms and pipelines is a very active field of research nowadays, broadly called *AutoML*. In broad terms we can consider that these methods apply a *wrapper approach*. They run the underlying algorithm/analysis pipeline on samples of the data (e.g. using cross-validation) to estimate the predictive performance of a given configuration, which is very different from our approach, in which we never run the full BioHEL algorithm during the hyper-parameter tuning process. A variety of strategies exist to search for the optimal configurations. For instance, TPOT [1] uses genetic programming to evolve trees that represent complete machine learning pipelines, including data cleaning, feature selection/construction and the selection and tuning of machine learning algorithms. ML-Plan [2] defines the algorithm tuning process as a planning task, and uses hierarchical planning networks [47] to identify the optimal plan, i.e. algorithm selection and tuning. A

different search strategy is employed by Auto-sklearn [4], which uses Bayesian optimisation for the *AutoML* task. As a final example we would like to mention *irace* [3] which, while intended to tune optimisation algorithms, uses an equivalent wrapper principle to the *AutoML* approaches. It uses an extension of the racing algorithm [48] to perform the tuning process. In the classic racing algorithm for machine learning model selection, a set of candidate models is evaluated sample by sample. In each step the methods that perform significantly worse are discarded. The process continues until a certain evaluation budget is reached or the set of remaining models is small enough.

Finally, another existing approach for algorithm selection (rather than tuning) is based on complexity measures defined to capture dataset difficulty in a supervised machine learning context [49]. These measures have been used to study the domains of competence of the XCS algorithm [50], as well as to recommend machine learning algorithms based on a meta-learning approach [51].

3 The BioHEL System

BioHEL [27] is an EML algorithm designed to handle large-scale datasets [52, 53, 54, 28, 29, 7, 8, 10]. BioHEL learns a set of rules following the Iterative Rule Learning paradigm first used in EML in the SIA system [55]. This learning paradigm generates ordered rule sets in which the rules in the solution are learnt sequentially, using a generational GA to learn individual rule. Hence, each individual of the GA population is a rule. Once a rule (the best individual in the final GA population) is learnt, it is added to the rule set and the training set is filtered by removing all the examples covered by this rule. The iterative process generally stops when the whole training set is covered. However, the stopping criteria changes when using a default rule as detailed in Section 3.1. In the rest of the section we only describe the aspects about BioHEL that are relevant to this work. For a full description of the algorithm, please see [27].

3.1 Representation

BioHEL uses the Attribute List Knowledge Representation (ALKR) [56], a sparse representation designed to handle high-dimensional problems. In this encoding, each rule represents only its relevant attributes, reducing considerably the cost of the match operations (as all the irrelevant attributes for that rule are not present). The relevant attributes can vary across rules, and are discovered during the learning process. ALKR uses hyper-rectangles [57] to represent continuous attributes and the GABIL representation [58] for nominal attributes.

Since in this paper we focus on problems with discrete (binary) attributes, it is necessary to explain the GABIL representation in greater detail. In this representation the attributes are expressed by binary strings of fixed length. The length corresponds to the number of possible values the attribute can have. For example, in a problem with three attributes (F_1, F_2 and F_3) if the attribute F_1 may take the values (A, B, C), F_2 the values (O, P), and F_3 the values (W, Z, X, Y) a possible condition string for each one of the attributes would look like:

F_1	F_2	F_3
100	01	1101

Each attribute is read as a disjunctive clause between all the values that have their bit on. For example, this condition can be interpreted as F_1 *is A and* F_2 *is P and* F_3 *is* $\{W$ *or* Z *or* $Y\}$.

To initialise a rule in ALKR first a subset of the problem's attributes is randomly chosen to be included in the rule. The size of the subset is controlled by the *ExpAtts* hyper-parameter.

Afterwards, an instance from the training set is sampled, and the bits corresponding to the instance's values in the GABIL predicate are set to 1. The rest of bits in the predicate are also set to 1 with probability p.

This representation uses an explicit default rule mechanism [59], which consists of a rule that covers all the examples left in the training set and assigns them to a user predefined class. Since the default class is not used in the evolved rules, this mechanism generates more compact rule sets. As a result of using a default rule, the stopping criteria of the iterative rule learning process is changed. In BioHEL, the rule learning process stops whenever it is not possible to generate a rule that has accuracy higher than the default rule.

3.2 Fitness Function

The BioHEL's fitness function is based on the Minimum Description Length principle [60]. This fitness function is designed to promote accurate, general and compact rules by integrating three metrics into the fitness formula: accuracy, coverage and complexity. This fitness function has two terms as shown in Equation (1).

$$F(a) = TL(a) \cdot W + EL(a). \tag{1}$$

$TL(a)$ (theory length) corresponds to the complexity of rule a, $EL(a)$ (exceptions length) corresponds to the accuracy and coverage of rule a and W is a weight that adjusts the relation between the previous terms. This hyper-parameter W is adjusted automatically using a heuristic defined by [59].

The definition of $TL(a)$ depends on the employed knowledge representation. For the GABIL representation it is defined as follows:

$$TL(a) = \frac{\sum_{i=1}^{NA} {}^{n_i}/_{v_i}}{NA_a}$$

where NA is the number of attributes in the problem, NA_a is the number of attributes explicitly represented by rule a, n_i is the number of values set to 0 and v_i is the number of possible values in the GABIL string for the i-th attribute represented in rule a. A simple interpretation of $TL(a)$ is that is computing the percentage of bits of the GABIL's encoding of a rule set to 0, with the added caveat that we use the ALKR sparse rule encoding in which only a faction of the attributes is represented, and only the represented attributes contribute to the formula. In this formula lower is better, so any non-represented attribute contributes 0 to the formula. Hence $TL(a)$ promotes rules in which (a) few attributes are represented and (b) few values within each attribute are set to 0.

Furthermore, $EL(a)$ is defined as:

$$EL(a) = 2 - ACC(a) - COV(a)$$

$$ACC(a) = \frac{correctlyClassified(a)}{matched(a)}$$

$$COV(a) = \begin{cases} 0 & \text{if } RC(a) < CB(c(a))/3 \\ MidCov(a) & \text{if } RC(a) < CB(c(a)) \\ HighCov(a) & \text{if } RC(a) \geq CB(c(a)) \end{cases} \tag{2}$$

$$MidCov(a) = CR \cdot \frac{RC(a)}{CB(c(a))}$$

$$HighCov(a) = CR + \frac{(1 - CR) \cdot (RC(a) - CB(c(a)))}{1 - RC(a)}$$

$$RC(a) = \frac{correctlyClassified(a)}{|T_{c(a)}|}$$

$$CB(c(a)) = CB \cdot \frac{|T|}{|T_c|}.$$

In these formulas, $ACC(a)$ corresponds to the accuracy of the rule and $COV(a)$ is the term that needs to promote general rules which is the key of BioHEL's fitness function. As shown in Equation (2), the value of $COV(a)$ depends on $RC(a)$ (recall; the ratio between the number of examples correctly classified by the rule over the total number of examples in the training set belonging to the same class as a) and $CB(c(a))$ (the percentage of examples of its class that any rule should cover to be considered a "good rule"). CB corresponds to the hyper-parameter known as *coverage breakpoint*. This hyper-parameter is first set globally and afterwards it is adjusted for every class in the problem based on the class distribution. This is a very problem dependent hyper-parameter which affects the performance of the system, as it was shown by [30] and the main target of the automated hyper-parameter setting of this paper.

Moreover, CR (coverage ratio) corresponds to the percentage of "reward" awarded to a rule with a higher coverage than the coverage breakpoint. Figure 1 shows the value of the coverage term $COV(a)$, depending on the coverage of the rule.

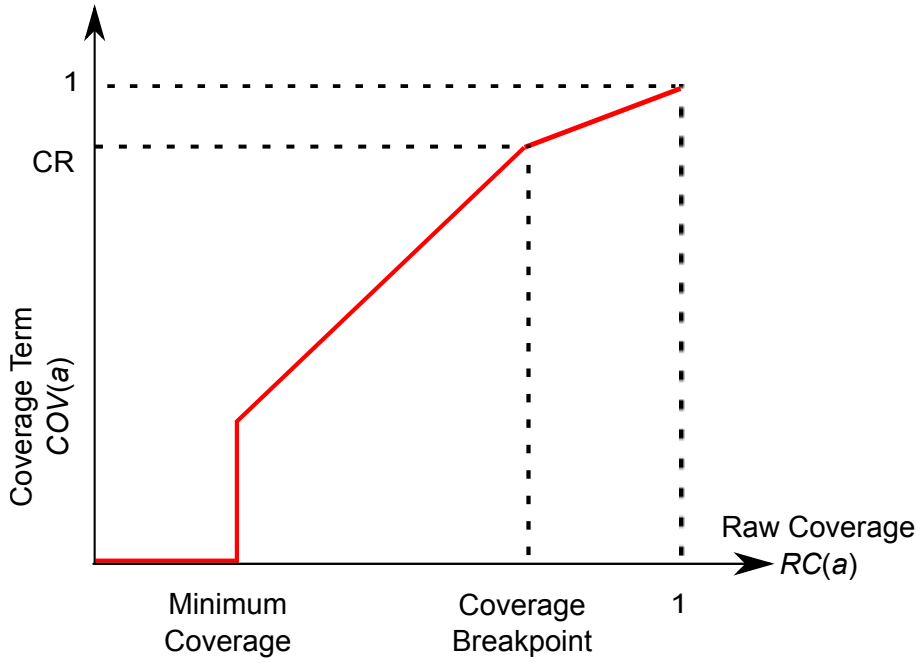


Figure 1: Coverage term $COV(a)$ according to rule coverage. The minimum coverage corresponds to one third of the coverage breakpoint

4 k-DNF Functions

k-Disjunctive Normal Form (k-DNF) functions [61] are a broad family of boolean functions which can be a useful tool to characterise the structure of binary problems. These functions

have been shown to be very useful to benchmark machine learning algorithms [62, 30, 63].

Given a space of d attributes or variables a k-DNF function is a boolean formula that presents the following form:

$$T_1 \vee T_2 \vee \cdots \vee T_r$$

where r is the number of disjunctive terms and each term T_x represents the conjunction of k boolean variables out of the d possible options $(x_1, x_2, \ldots, x_d)$, where some of the variables might have the *not* function ($\neg$) applied to them. Equation (3) represents an example of a k-DNF function for a space of 10 representable attributes (d), 2 terms (r) and 4 represented attributes (k).

$$(x_1 \wedge x_5 \wedge \neg x_7 \wedge x_{10}) \vee (x_1 \wedge \neg x_3 \wedge \neg x_6 \wedge x_7). \tag{3}$$

To construct a machine learning problem from a k-DNF boolean formula we generate all the possible 2^d instances (binary strings of size d). Afterwards, if the formula holds for a particular instance (one or more terms are true), class 1 (i.e. the positive class) is assigned to it. Otherwise class 0 (the negative class, to be covered by the default rule) is assigned. Considering this, the optimal rules that correspond to the solution of the previous problem in Equation (3)[1] would be:

```
1###1#0##1 : 1
1#0##01### : 1
Default Rule : 0
```

Since most EML systems learn a set of rules as the solution of a problem, k-DNF formulas are really useful to evaluate them, as the systems are expected to learn one rule for each of the r terms in the problem and correct rules should represent at least the k relevant attributes in these terms. From this point onwards, every time we talk about "rules" we are referring to the solution of the problem and when we talk about "terms" we refer to the problem itself.

Many well-known boolean benchmarks can be represented by a function in k-DNF [30]. For instance, the rules in the 6-bit multiplexer have a k of 3 (two address bits and one data bit), the rules in the 20-bit multiplexer have a k of 5, a k of 6 is present in the 37-bit multiplexer and so on. The 18-bit hybrid Parity-Multiplexer problem [22] has a k of 9, as this problem is composed by a 6-bit multiplexer where each of these "bits" is the result of a 3-bit parity problem.

The difficulty of the k-DNF problems is closely related to the class imbalance. The class imbalance of a k-DNF problem can be estimated by calculating the probability of finding a negative example in the dataset given specific values for k and r.

$$P(neg) = \left(1 - 2^{-k}\right)^r. \tag{4}$$

Considering each term covers a percentage of 2^{-k} of the training set, this formula states that the probability of having a negative example is equal to the probability of the example not being covered by any of the r terms in the problem. This formula holds under the assumption that the k attributes of the r terms are randomly picked and hence there is no large amount of overlap between rules.

Figure 2 shows the corresponding probability distribution for $P(neg)$. Here we can see that, depending on k and r, scenarios with very high class imbalance can be possible. When k is low each term covers a large proportion of the training set, and hence, just with a few terms

[1]For simplicity we present the rules in ternary representation, where # means the attribute is irrelevant, and 0 or 1 represent the value the attribute should take to make the predicate hold. However, BioHEL uses GABIL to represent binary and discrete attributes as shown in Section 3.1.

(r), most of the examples will be positive. In these situations we encounter a known source of difficulty for EML systems: term overlap [31, 64]. On the other hand, a high k and low r create problems with very few positive examples, as each term is very specific and overlap is unlikely. These cases, essentially become scenarios of trying to find the needle in the haystack. Since the class imbalance makes the problem more difficult, the red area represents the problems that are easier to solve. For more information about the generation of artificial k-DNF problems please refer to [30].

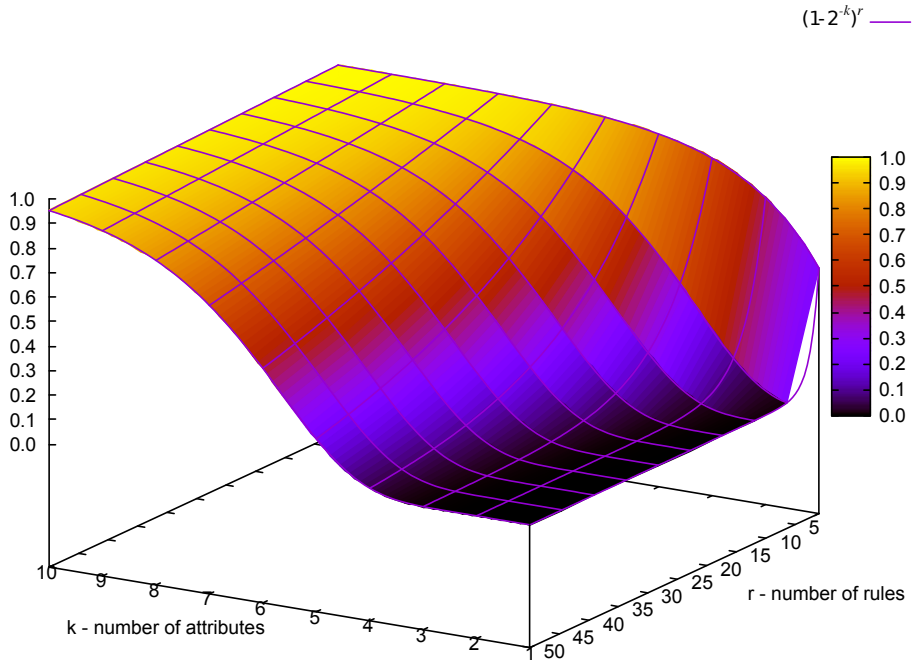


Figure 2: Probability of having a negative example in a k-DNF function according to the number of attributes k and the number of terms r

5 Automatic Hyper-Parameter Setting

Considering the importance of the coverage breakpoint hyper-parameter in the performance of the BioHEL system, it seems necessary to adjust this hyper-parameter automatically for several reasons:

- **Runtime.** Since BioHEL is a system mainly oriented to solve large scale datasets, finding the correct setting for problem dependant hyper-parameters such as the coverage breakpoint involves a time-consuming preliminary experimentation stage. The automatic setup of this hyper-parameter can avoid preliminary experimentation and reduce the total experimental time.

- **Usability.** In many cases, end-users avoid exhaustive experimentation and settle for naive configurations that do not produce the best results. This improvement could make the system easier to use to an end-user and could also find better solutions for problems where the adequate coverage breakpoint has not been determined yet.

According to [30] it is possible to determine the coverage breakpoint if the characteristics of the problem (k-DNF formula) are known. These characteristics are the number of attributes expressed in the terms k and the total number of terms in the formula r. According to this work if the number of attributes in the terms is k the adequate coverage breakpoint to solve the problem is 2^{-k}. To ensure learning, the coverage breakpoint should be equal or smaller than this value. This translates the problem of finding the adequate coverage breakpoint to finding k.

But how it is possible to identify the structure of the problem by observing the data? To do this it is necessary to develop models based on k and r that explain characteristics of the data and behaviours of the system when working this data. For example, the model for the number of negative examples in Equation (4) finds a correlation between k and r and the proportion of negative examples in the problem. Moreover, the probability of obtaining good individuals, modelled for BioHEL using the ALKR representation and the GABIL encoding [31], also provides a relationship between a characteristic of the initial population and the variables k and r. This means that by observing these characteristics we can have estimates of the k and r of the problem.

However, these models only indicate a relationship between k and r. Many different values for k and r can satisfy the equation and, independently, each model does not provide information useful enough to identify these problem characteristics. However, by combining the results of different models together it is possible to determine the values for k and r that are more likely to match the characteristics of the problem.

In this paper we present a heuristic approach to determine the k and r of a given unknown boolean problem at runtime. This approach works by classifying the problems based on observations made over the data and randomly sampled individuals. Using this information the problems are classified into groups with a particular k and r associated, which we will call *kr-groups*.

The classification is done using a voting system. The space of k and r is divided uniformly in *kr-groups* which have an associated expected value and standard deviation boundaries for each one of the characteristics we measure. When a problem presents a characteristic that falls into the standard deviation boundaries of a particular *kr-group* the group gets awarded points. At the end, the *kr-group* that obtained more votes is considered the winner. Figure 3 illustrates how the heuristic works.

Particularly for BioHEL, three characteristics were considered to classify the problems:

- The number of negative examples in the problem (defined in section 5.1.1).
- The number of *good individuals* in a random sample after evaluating them against the given problem. These are individuals that do not make classification mistakes or that have an accuracy higher than a certain threshold (defined in section 5.1.2).
- The number of attributes expressed in the good individuals (defined in section 5.1.3).

The following sections will explain in greater detail each one of the criteria used to classify the problems (section 5.1). Afterwards, we will show in more detail how the *kr-space* is partitioned in *kr-groups* and what makes a problem belong to a specific group (section 5.2). Finally, we explain the algorithm step-by-step (section 5.3).

5.1 Classification Criteria

This section explains each one of the criteria used within BioHEL to classify the problems: a) the number of negative examples in the problem, b) the number of *good individuals* in a random sample after evaluating them against the given problem and c) the number of attributes

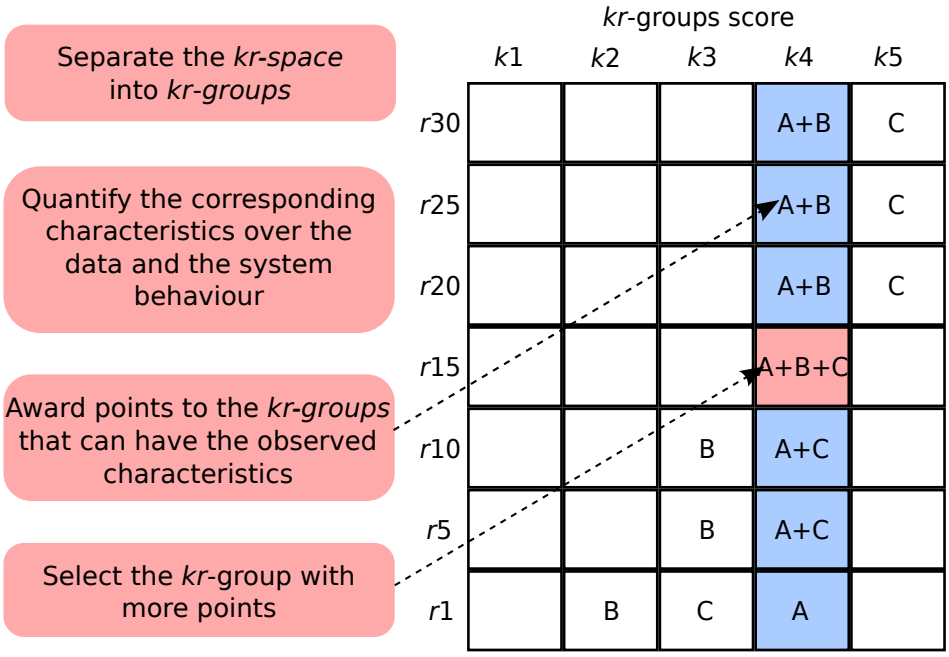


Figure 3: High-level representation of the problem structure identification heuristic. A, B and C refer to the criteria that will be used to assign points to the $kr-groups$

expressed in the good individuals. The first two characteristics used are completely theory-driven, which means they use theoretical models to determine the kind of problem we are handling. The last characteristic, even though it does not come from a model, reinforces the two previous criteria in finding the correct *kr-group*. Since the good individuals are already calculated for the second criterion, using the number of attributes expressed in these individuals does not involve an extra computational cost.

5.1.1 Number of negative examples in the problem

Depending on the number of terms r and number of attributes expressed in each term k, the k-DNF problem will present a different percentage of negative examples.

For a randomly generated binary problem defined as the disjunction of r terms, where each term is the conjunction of k randomly picked attributes, the probability of having a negative example in the training set $P(neg)_r^k$ is equal to Equation (4) shown in Section 4.

By counting the number of negative examples in the training set, it is possible to use this formula inversely to determine possible combinations of k and r that are feasible for the given problem. For example a problem with $k = 2$ and $r = 1$ has 75% of negative examples. But also a problem with $k = 6$ and $r = 18$ has on average the same percentage of negative examples. If we observe a particular problem with 75% of negative examples both of these *kr-groups* would receive scores according to this criterion.

5.1.2 Number of good individuals in a randomly initialised sample

A good individual or a *representative*, as it was defined by [22], is a rule that specifies (has represented) correctly at least all the attributes in one of the terms of the optimal solution to the problem. For example, if one of the terms of a problem with $d = 5$ and $k = 2$ is $x_1 = 0 \wedge x_4 = 1$ (0**1*) possible representatives would be 0##11, 0##1# and 01110, where # means that the attribute can take any value. Therefore, this rule does not make mistakes, but it can be more specific than the optimal rule where only k attributes are specified. The probabilities of finding a representative were first proposed by [22] for the ternary representation $\{0, 1, \#\}$. However,

these models were not entirely suitable for BioHEL, as this system uses a different encoding. Later on, suitable models for the binary domain using the ALKR+GABIL representation were proposed by [31] .

Assuming the usage of the default rule and covering mechanisms, the probability of finding a representative for a binary problem depends on k and r, as shown in Equation (5). This function states that the probability of having a good classifier $P(rep)$ is equal to the probability of having at least one of the terms in the k-DNF problem represented, and to have a term represented the rule should express the k relevant attributes. These models are able to hold with a certain amount of rule overlap if the rule coverage is uniform. For more details about this model please see full description presented by [31].

$$P(rep) = 1 - \left(1 - \left(\frac{2^{-k}\left(l_d(1-p)\right)^k}{1-(1-2^{-k})^r}\right)\right)^r . \tag{5}$$

In this formula p corresponds to the probability of setting to 1 the values in a GABIL attribute (see Section 3), and l_d is the probability that an attribute appears in the ALKR attribute list. This value at the same time depends on the user-defined hyper-parameter $ExpAtts$ (expected number of attributes) as follows:

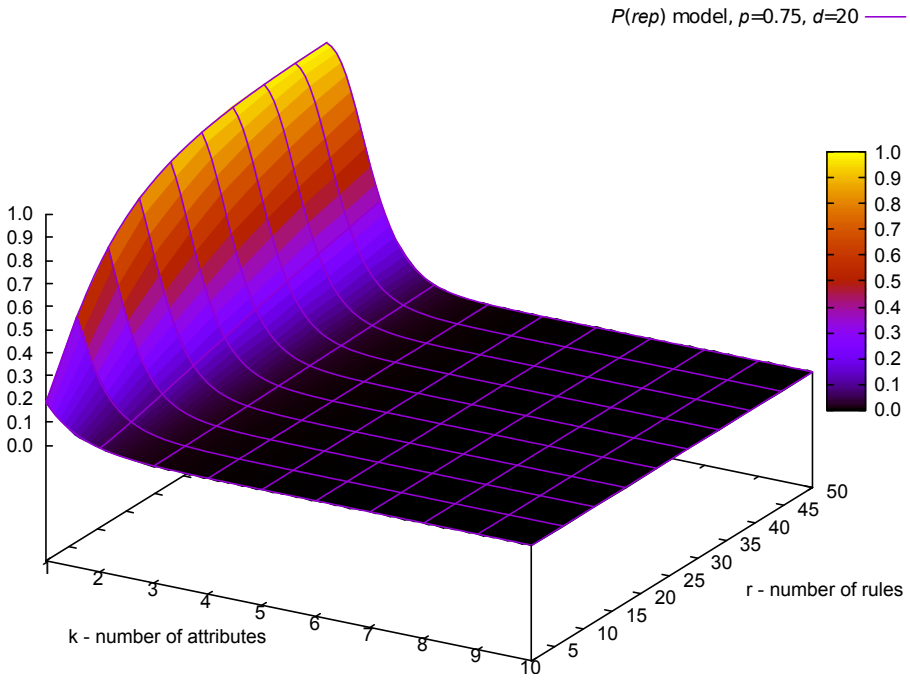
$$l_d = \begin{cases} 1 & d <= ExpAtts \\ \frac{ExpAtts}{d} & d > ExpAtts \end{cases} .$$

Figure 4 shows an example of the landscape of this model using different values of p. By counting how many representatives are found in a randomly initialised sample of individuals it is possible to use the formula inversely to determine feasible pairs of k and r for the given problem.

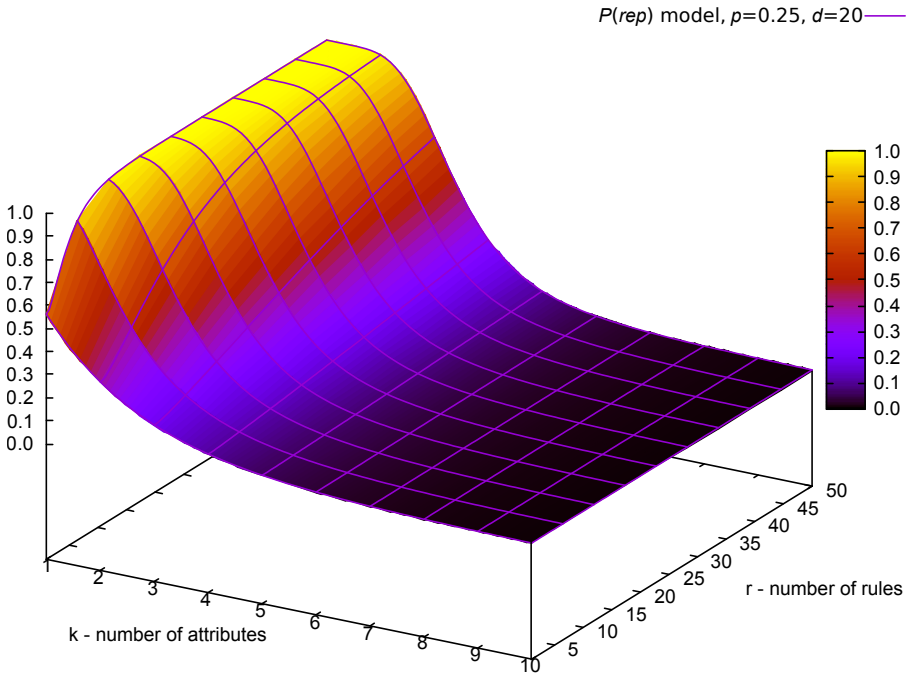
The individuals for the sample are not generated one by one, but by chunks of N individuals (for all experiments in this paper N=500). After evaluating N rules, it might be possible that we do not find any representatives. This could happen due to several reasons. Either the sample is too small and/or the probability of a representative for a particular point is too small as well. To solve these problems the system increases iteratively the total sample size (generates N additional samples) until R representatives (hyper-parameter set by the user) are found. This guarantees that the number of representatives found is not zero while checking the lowest number of individuals as possible. A high value of R will involve checking a bigger sample size, while a small value would have the opposite effect.

Moreover, we try to generate R representatives using the largest value of p possible, because that would create more general rules, as shown in Figure 4. However, more general random rules are more likely to make mistakes. Therefore, when the problem has a larger k, smaller values of p are needed to generate rules that do not make mistakes. The procedure that the system follows to adjust p, while finding the representatives, is shown in Algorithm 5.1. Within this algorithm, *genSample* initialises N rules following the procedure explained in section 3.1. Moreover, *getAccuracy* computes the accuracy of a rule across the training set. Finally, getMostFrequentK simply identifies the k value most frequently used within the set R of representatives generated by the heuristic.

The system first tries to obtain representatives generating populations of size N created using the largest value of p: $pmax$. Then all these individuals are evaluated against the training set. Afterwards, all the rules with accuracy higher or equal than $minAcc$ are considered representatives. When R or more representatives are found the system returns the representatives found. If the system has already checked 6 samples and has not found any representatives, the value of p is lowered globally across the system and the search continues. If the value of p has reached its minimum value and the system has not found representatives yet, the search aborts.



(a) p=0.75



(b) p=0.25

Figure 4: Probability of generating a representative with different values of p in a problem with $d = 20$ and $ExpAtts = 15$.

The calculation of representatives from a given sample is interesting because it gives room for our third criterion, which is the number of attributes observed in them. However, the identification of the genuine representatives is far from trivial, and some post-processing is

Algorithm 5.1: SEARCHREPS(N)

```
p ← pmax
while p ≥ pmin
  do { rep ← ∅
       while i < 6 ∨ rep ≠ ∅
         do { sample ← GENSAMPLE(N, p)
              for c ∈ sample
                do { if GETACCURACY(c) ≥ minAcc
                       then { c ← PRUNING(c)
                              rep ← rep ∪ c
              if |rep| ≥ R
                then { k* = GETMOSTFREQUENTK(rep)
                       for c ∈ rep
                         do { if |c.atts| ≠ k*
                                then rep = rep − c
                       return (rep)
              i = i + 1
       p = p − pstep
return (null)
```

Algorithm 5.2: PRUNING($Classifier\ c1$)

```
prevacc ← GETACCURACY(c1)
for each att ∈ GETATTRIBUTES(c1)
  do { REMOVEATTRIBUTE(att, c1)
       if GETACCURACY(c1) >= prevacc
         then prevacc ← GETACCURACY(c1)
         else RESTORE(att, c1)
return (c1)
```

needed as it will be explained in the next section.

5.1.3 Number of attributes in the representatives

According to the definition of a representative the number of relevant attributes in a candidate representative cannot be less than k, otherwise this rule would make mistakes. This gives us at least an upper bound of the problem's k. However, in order to use the characteristics of the representatives we need to make sure that these "good rules" are actually genuine representatives and that they have the minimum possible amount of attributes without making mistakes.

Two problems might arise with the good rules. First, it can happen that the good rules have more attributes specified in the actual terms of the problem, which is misleading. Second, when the problem consists in more than one rule, the system might find classifiers that do not make any mistakes but they do not represent the terms of the problem. These classifiers instead

represent the union or intersection between two or more terms. They might have more or less attributes expressed than k, and they are not really representatives of the problem. Therefore, in order to find genuine representatives it is necessary to post-process the sample rules.

To tackle the first problem we need to eliminate the attributes that do not affect the accuracy. As shown in Algorithm 5.1 this is done right before adding the rule to the representative set. To prune unnecessary attributes we perform an iterative search process as shown in Algorithm 5.2. Each one of the attributes in the rule is eliminated, one by one. If the accuracy decreases, the classifier is restored, if not the search continues over the resulting classifier. This local search operator was already proposed by [65] as a post-processing operator to refine the generality of the rules.

To tackle the second problem we need to eliminate the deceptive representatives. That is, the classifiers that do not make mistakes but do not really correspond to the terms of the problem we want to learn. Since these rules usually have a number of attributes either larger or smaller than k (but not exactly k), we keep only the ones that correspond to the most frequent number of attributes observed k^* in the set of R representatives. In the end the rules left in the set are considered genuine representatives and they are the input for the second criterion (Section 5.1.2). Moreover, as we have already calculated the most frequent number of attributes observed among the representatives, we can also use this information k^* as a third metric to award points to the *kr-groups* with $k = k^*$.

5.2 Classifying the Problems

As we explained before, to use the model it is necessary to calculate the standard deviation boundaries for each one of the possible combinations of k and r. For this, we sample uniformly the $kr - space$ and we calculate the expected value, the lower bound and the upper bound for each point. To sample the space we calculate these values for $k = \{1..d\}$ and $r = \{1..100\}$ using a step size of 5 for the rules.

Moreover, to calculate the lower and upper bounds for each point we need to analyse further the probabilistic models used. For the probability of a representative we can consider that the probability of having a specific number of representatives in a sample of N classifiers follows a binomial distribution with probability $P(rep)$. This assumption comes from the fact that the generated rules have the same probability of becoming representatives and they are independent from each other. Therefore, the probability of having x representatives can be written as follows:

$$P(\#rep = x) = \binom{N}{x}(1 - P(rep))^{N-x}(P(rep))^{x}.$$

In this case we know that for each point the mean percentage of representatives is $P(rep)$ and the variance is $var = (P(rep)) * (1 - P(rep))$. So for a problem to belong to a specific $kr - group$, we check if the empirical percentage of representatives observed $P'(rep)$ is between the boundaries as follows:

$$P(rep)_r^k - var \leq P'(rep) \leq P(rep)_r^k + var.$$

In the case of the class imbalance we actually do not know the probability distribution, but we know the mean value given k and r. We cannot assume that it is a binomial distribution because the examples in a training set are not independent observations (usually they are not repeated). In this case fixed intervals are used to determine if a problem belongs to a certain group. To do this we calculate the empirical value $P'(neg) \pm 0.1$ to classify the problems as follows:

$$P(neg)_r^k - 0.1 \leq P'(neg) \leq P(neg)_r^k + 0.1.$$

5.3 Hyper-Parameter Setting Procedure Step-by-step

To recapitulate and explain better how the concepts and methods presented are merged together to produce our approach, this section will explain step-by-step the algorithm used within BioHEL to determine the k and r of a problem and its corresponding coverage breakpoint. The algorithm consists in the following steps also shown graphically in Figure 5.

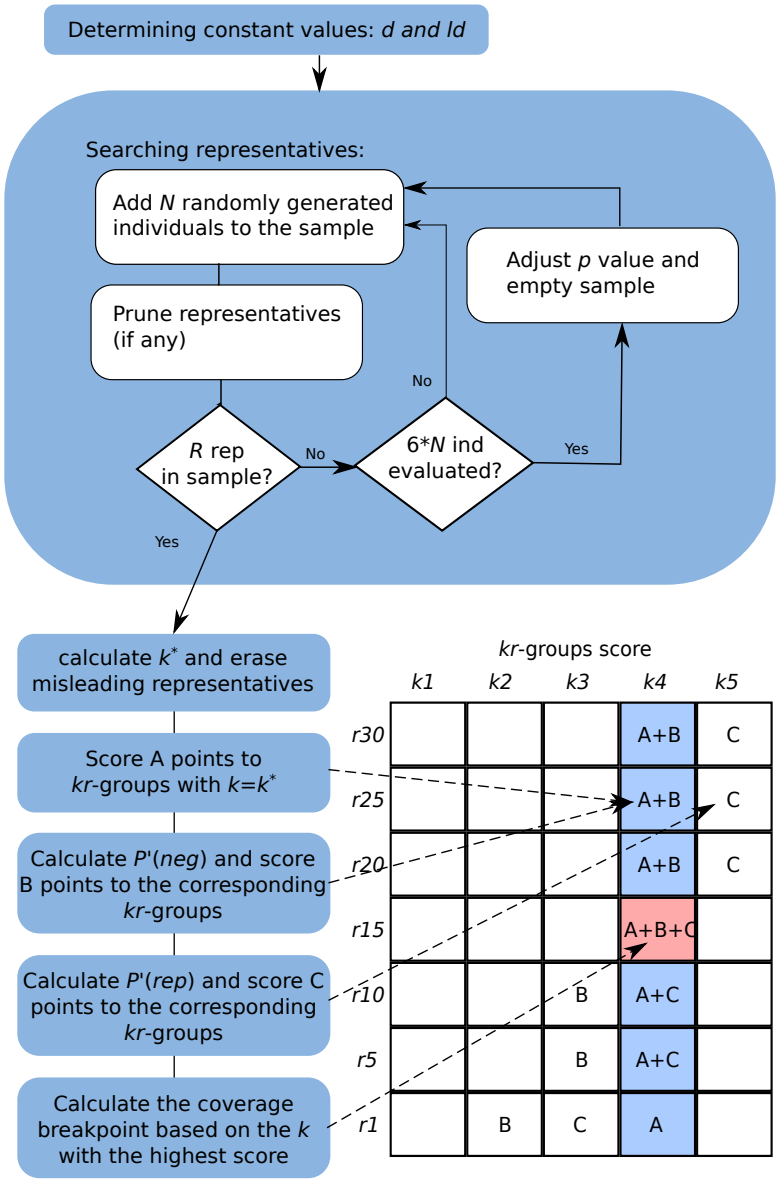


Figure 5: Steps to find the adequate coverage breakpoint with and example of the final score grid.

1. Determining the number of attributes in the problem d and the value l_d.
2. Searching R representatives in a randomly initialised sample. Finding representatives involves the following sub-steps:
   (a) **Representative generation** Searching iteratively in population of N classifiers until a total of R representatives is found, by evaluating them across the whole training set.

(b) **Representative pruning.** When a good rule is found, the system removes all the attributes that can be eliminated without degrading the accuracy.

(c) **Adjustment of initialisation hyper-parameters.** If the system has checked already 6 populations of size N and have not yet found any representatives, the system re-adjusts the value p (See Section 5.1.2).

3. Calculating the most frequent number of attributes activated in the candidate representatives and erasing the misleading ones, keeping only the ones that have a k equal to the most frequent value observed k^*.

4. Determining the number of examples in the training set belonging to the default class ($P'(neg)$).

5. Calculating the observed value $P'(rep)$ as the number of representatives observed divided by the total number of rules observed (total sample size).

6. Calculating the score of a *kr-group* ($Score_r^k$) with equation (6) where $k == k^*$, Neg_r^k and Rep_r^k are boolean variables that take the value of 1 if the empirical observation matches the criteria of the *kr-group* and A, B and C are weights associated to each of the three criteria of the heuristic.

$$\begin{aligned} Score_r^k &= A \cdot (k == k^*) + B \cdot Neg_r^k + C \cdot Rep_r^k \\ Neg_r^k &= P(neg)_r^k - 0.1 \leq P'(neg) \leq P(neg)_r^k + 0.1 \\ Rep_r^k &= P(rep)_r^k - var \leq P'(rep) \leq P(rep)_r^k + var. \end{aligned} \tag{6}$$

7. To finalise, calculating which is the smallest group (smallest k and r) that obtained the highest coincidences between the 3 metrics (highest score). This k is transformed in the coverage breakpoint as $CB = 2^{-k}$.

6 Experimental Design and Results

In this section we present the experimental framework used to test our approach and the corresponding results. First we analyse the hyper-parameter setting approach over a wide variety of synthetic k-DNF problems, with and without noise, created using the generator we provide in `http://ico2s.org/datasets/kdnf.html`. Then we present an additional test of our approach over a binary protein structure prediction problem already used by [52] which constitutes an interesting challenge for our approach due to the high levels of noise found in the problem. The code and all the datasets used for these experiments can be found in `http://ico2s.org/data/instances/cov-break-heu/`.

6.1 Analysis of the Hyper-Parameter Setting Approach Over Binary Problems

In this section we analyse the performance of our approach over a wide variety of k-DNF problems, in terms of probability of success (finding the adequate hyper-parameter value). At the end, we also comment briefly on the additional effort incurred by the heuristic in terms of additional evaluation operations.

The k-DNF problems used in this section have the following characteristics: $d = 20$, $k = \{2 - 9\}$, and $r = \{5, 10, 20, 40\}$. Moreover, we introduced output noise of 0%, 1%, 5% and 10% over the problems to determine how robust was the classification process towards noise.

Table 1: Hyper-parameters for the heuristic used to characterise and find the coverage breakpoint for k-DNF problems.

Hyper-parameter	Value
Number of representatives needed - R	10
Evaluated pops to change p	6
Most frequent k in representatives - Score A	2
Imbalance function - Score B	2
Prep function - Score C	1
Sample size - N	500

We generated 5 different problems of each k-DNF scenario, and each problem was run with 5 different seeds. Also, all these runs were performed using fixed default class 0. Since in the k-DNF problems all the generated terms map to class 1, this setting prevents the system form learning the inverse problem, over which calculating the success would not be straightforward.

The learning process was not performed during this stage of experiments, but only the hyper-parameter setting stage. In these experiments, we want to quantify how many times the heuristic finds the optimal k for the problem (or at least a larger one) which would ensure the learning.

We also experiment changing the hyper-parameter *minAcc* (the minimum accuracy demanded in a rule to become a representative) to determine how this hyper-parameter affects the search, and show how it can help tackling problems with noise more efficiently. In these experiments we tested hyper-parameter values $minAcc = \{1.0, 0.95, 0.9\}$. To determine significant differences among using different *minAcc* values we used a Friedman test with its post-hoc Holm test, as shown by [66].

The rest of the hyper-parameters in our approach are shown in Table 1 for clarity and replication purposes. However, according to our preliminary experiments, the hyper-parameters shown in this table can be considered constants and they can remain fixed. Only the minimum accuracy *minAcc* and the number of representatives R have an important impact on the results, because they are directly related to the problem noise and the additional search effort, respectively. Analysing R in depth would require a very long and complex experimentation. For simplicity in this paper we have set the hyper-parameter to a value (10) that in preliminary work showed to be suitable for all tested scenarios, although a smaller R would also work in some of the easier datasets. Moreover, it should be noted that the reason why the $P(rep)$ function has a score lower than the two other metrics is because, in preliminary experimentation, this metric was not as reliable as the other ones.

6.1.1 Results

Table 2 presents the results for the different k-DNF configurations and different values of *minAcc* in terms of percentage of success (finding the k of the problem or at least a larger one). Results for $k = 2$ and $r = 40$ are omitted because in these setting all instances belong to class 1. The cells emphasised represent the configurations where the success rate is less than 100%. For $minAcc < 1$, the cells marked with red show the cases where the success rate is lower than the base case ($minAcc = 1.0$), and the cells marked with green show the cases where the success rate increased. In this table we can observe that using $minAcc = 1$ the heuristic is able to find the appropriate coverage breakpoint for most of the configurations with no noise. Moreover, it is noticeable that the output noise affects the performance of the heuristic.

On the other hand, we can observe that the heuristic fails in the cases where there is very

Table 2: Probability of success in k-DNF problems with different amounts of output noise and minimum accuracy thresholds ($minAcc$). Cells in bold emphasise the cases where the heuristic was not 100% successful. Cells in green and red emphasise the cases where using a lower $minAcc$ value obtained better and worst results, respectively.

		$minAcc$=1.0				$minAcc$=0.95				$minAcc$=0.9			
Prob d=20		Noise Level				Noise Level				Noise Level			
k	r	0%	1%	5%	10%	0%	1%	5%	10%	0%	1%	5%	10%
2	5	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	**72.00**	100.00	100.00	100.00
2	10	100.00	**0.00**	**0.00**	100.00	**32.00**	**64.00**	100.00	100.00	**0.00**	**4.00**	**20.00**	100.00
2	20	**80.00**	**4.00**	**0.00**	**0.00**	**32.00**	**0.00**	100.00	**0.00**	**32.00**	**0.00**	**24.00**	100.00
2	40		**0.00**	**0.00**	**0.00**		**0.00**	**80.00**	**0.00**	**0.00**	**4.00**	**0.00**	**96.00**
3	5	100.00	100.00	**24.00**	**0.00**	100.00	100.00	100.00	**4.00**	**96.00**	100.00	**88.00**	100.00
3	10	100.00	**84.00**	**44.00**	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00
3	20	100.00	**0.00**	**0.00**	**0.00**	**16.00**	**80.00**	100.00	**0.00**	**0.00**	**0.00**	**12.00**	100.00
3	40	100.00	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	100.00	**0.00**	**0.00**	**0.00**	**0.00**	100.00
4	5	100.00	100.00	**68.00**	**4.00**	100.00	100.00	100.00	**28.00**	100.00	100.00	100.00	100.00
4	10	100.00	**96.00**	**16.00**	**0.00**	100.00	100.00	100.00	**0.00**	100.00	100.00	100.00	100.00
4	20	100.00	**4.00**	**0.00**	**60.00**	100.00	100.00	100.00	**60.00**	**92.00**	100.00	100.00	100.00
4	40	100.00	**0.00**	**0.00**	**0.00**	**4.00**	**24.00**	100.00	**0.00**	**0.00**	**0.00**	**4.00**	100.00
5	5	100.00	100.00	100.00	**44.00**	100.00	100.00	100.00	**44.00**	100.00	100.00	100.00	100.00
5	10	100.00	100.00	**64.00**	**0.00**	100.00	100.00	100.00	**8.00**	100.00	100.00	100.00	100.00
5	20	100.00	**60.00**	**8.00**	**0.00**	100.00	100.00	100.00	**0.00**	100.00	100.00	100.00	100.00
5	40	100.00	**4.00**	**0.00**	**0.00**	100.00	100.00	100.00	**0.00**	**92.00**	**80.00**	100.00	100.00
6	5	100.00	100.00	100.00	**60.00**	100.00	100.00	100.00	**88.00**	100.00	100.00	100.00	100.00
6	10	100.00	100.00	100.00	**28.00**	100.00	100.00	100.00	**8.00**	100.00	100.00	100.00	100.00
6	20	100.00	100.00	**48.00**	**0.00**	100.00	100.00	100.00	**0.00**	100.00	100.00	100.00	100.00
6	40	100.00	**40.00**	**0.00**	**0.00**	100.00	100.00	**80.00**	**0.00**	100.00	100.00	100.00	**84.00**
7	5	100.00	100.00	100.00	**88.00**	100.00	100.00	100.00	**92.00**	100.00	100.00	100.00	100.00
7	10	100.00	100.00	100.00	**28.00**	100.00	100.00	100.00	**48.00**	100.00	100.00	100.00	100.00
7	20	100.00	**88.00**	100.00	**4.00**	100.00	100.00	100.00	**16.00**	100.00	100.00	100.00	**92.00**
7	40	100.00	**76.00**	**4.00**	**0.00**	100.00	100.00	**92.00**	**0.00**	100.00	100.00	100.00	**76.00**
8	5	100.00	100.00	100.00	**80.00**	100.00	100.00	100.00	**76.00**	100.00	100.00	100.00	**96.00**
8	10	100.00	100.00	100.00	**68.00**	100.00	100.00	100.00	**72.00**	100.00	100.00	100.00	100.00
8	20	100.00	100.00	100.00	**4.00**	100.00	100.00	100.00	**12.00**	100.00	100.00	100.00	**80.00**
8	40	100.00	**84.00**	**92.00**	**0.00**	100.00	**84.00**	100.00	**0.00**	100.00	**84.00**	100.00	**44.00**
9	5	100.00	100.00	**84.00**	**48.00**	100.00	100.00	**92.00**	**24.00**	100.00	100.00	**88.00**	**68.00**
9	10	100.00	100.00	**96.00**	**68.00**	100.00	100.00	100.00	**72.00**	100.00	100.00	100.00	**84.00**
9	20	100.00	100.00	100.00	**36.00**	100.00	100.00	100.00	**32.00**	100.00	100.00	100.00	**92.00**
9	40	100.00	100.00	**80.00**	**4.00**	100.00	100.00	**80.00**	**8.00**	100.00	100.00	**72.00**	**44.00**

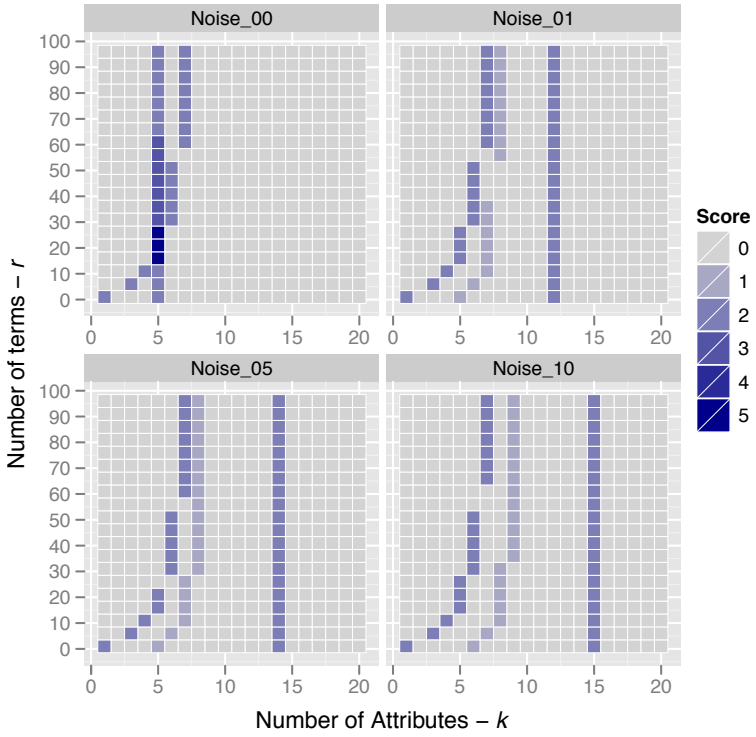


Figure 6: Final score grids of the heuristic using $minAcc = 1$ in a problem with $d = 20$, $k = 5$ and $r = 20$ for the 4 levels of noise.

high rule overlapping. These problems are very difficult to solve by the system because of the class imbalance [30], so it is not surprising that they are difficult for the heuristic as well. Such large overlapping makes the heuristic think that the k is smaller than the real one. In the case of a synthetic problem like the k-DNF, where we know the correct answer, this is incorrect. However, what the system is trying to do is not completely wrong, because it is trying to solve the problem with less complex rules compromising the accuracy slightly, which in real-life domains can be advantageous. To understand better these domains further research focusing specifically on the rule overlapping scenario is required.

Figure 6 shows an example of the final score grid of the heuristic using $minAcc = 1$ for a problem with $k = 5$ and $r = 20$ with the 4 levels of noise. In each of the four plots (for a different level of noise) the vertical stripe corresponds to the 2 points that are awarded to the most frequent number of attributes observed in the representatives. The curved stripes correspond to the scores awarded by the other two criteria of the heuristic. In this figure we can see that while the problem increases the representatives present a higher number of attributes. When this happens this area does not intersect the two other areas, thus the heuristic fails to find the appropriate k value. This is because the constraint of $minAcc = 1.0$ is too strict in these cases where the problem has noise. In the cases with noise we should take in consideration that good rules will have a good accuracy, but not equal to 1. Relaxing this hyper-parameter, as we can see in Table 2, helps finding adequate representatives for problems with noise. We can observe here that the success rate increases in most cases, except when the problem has no noise. As we expected to observe, when the problem has 5% noise, the best results are obtained using $minAcc = 0.95$, and the same occurs when the problem has 10% noise and we use $minAcc = 0.9$.

Moreover, Table 3 contains the statistical analysis to determine which value of $minAcc$ is best for the different sets of problems, distinguishing them by the amount of noise. In this table we can observe that the best method changes as expected, and for the problems with high

Table 3: Results of the Friedman test performed to determine the best *minAcc* value depending on the noise. *Control* shows which was the algorithm that obtained the best ranking. *Dominance* shows the configurations that are significantly worst than the control configuration according to the Holm post-hoc test with confidence threshold 0.05.

			Control			
d	Noise	p-value	1.0	0.95	0.9	Dominance
20	0%	0.00050	⋆			0.9
20	1%	0.00100		⋆		1.0
20	5%	1.895e-07		⋆		1.0, 0.9
20	10%	1.119e-11			⋆	1.0, 0.95

amount of noise (5% and 10%) the respective best method ($minAcc = 0.95$, $minAcc = 0.90$) performs better than the rest of the configurations.

Based on these results we can state that for problems with noise we should use a minimum accuracy equal to the percentage of noise observed in the problem. Even though this introduces a new hyper-parameter *minAcc*, the percentage of permitted noise is a much more intuitive hyper-parameter to set up than the coverage breakpoint, since it is an structural hyper-parameter of the problem, instead of being a hyper-parameter of the system.

One interesting aspect to notice in Table 2 is that for problems with larger k, the heuristic seems to fail when the problem has either too many or too few terms. When the problem has too many terms, it is possible to find representatives but these representatives are likely to have a large k and this value might not intersect with the other two areas, as it was exemplified in Figure 6. Moreover, if the problem has few terms finding a representative becomes very difficult and it is possible that the system does not find any representatives during the search process. In this case the mechanism will only rely on the imbalance metric to make a decision.

Our hypothesis is that these difficulties can be tackled by changing the selection policy of the k when there is not a single cell where the three metrics have intersected. Moreover, adjusting the minimum accuracy along the search process or a more granular step size in the adaptation of p could help obtain better results by finding representatives when this task is very difficult. More experimentation is needed to validate these hypotheses.

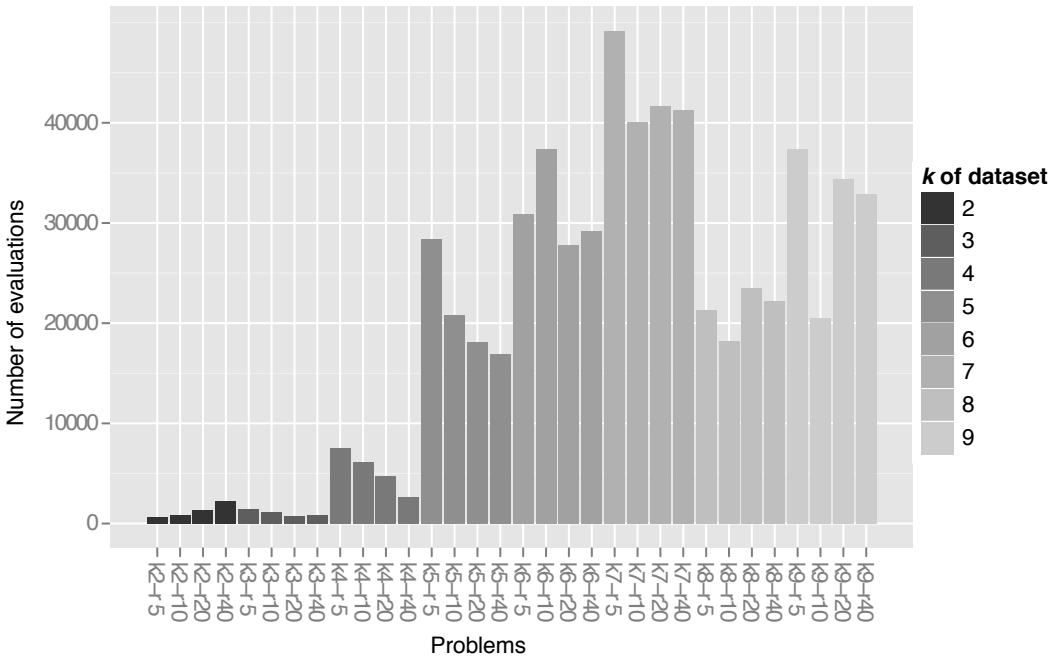
Based on these results, we can conclude that our hyper-parameter setting mechanism is able to find the appropriate k value for a wide variety of binary problems, including problems with noise. We explore next the computational overhead of our approach.

6.1.2 Computational effort of the heuristic

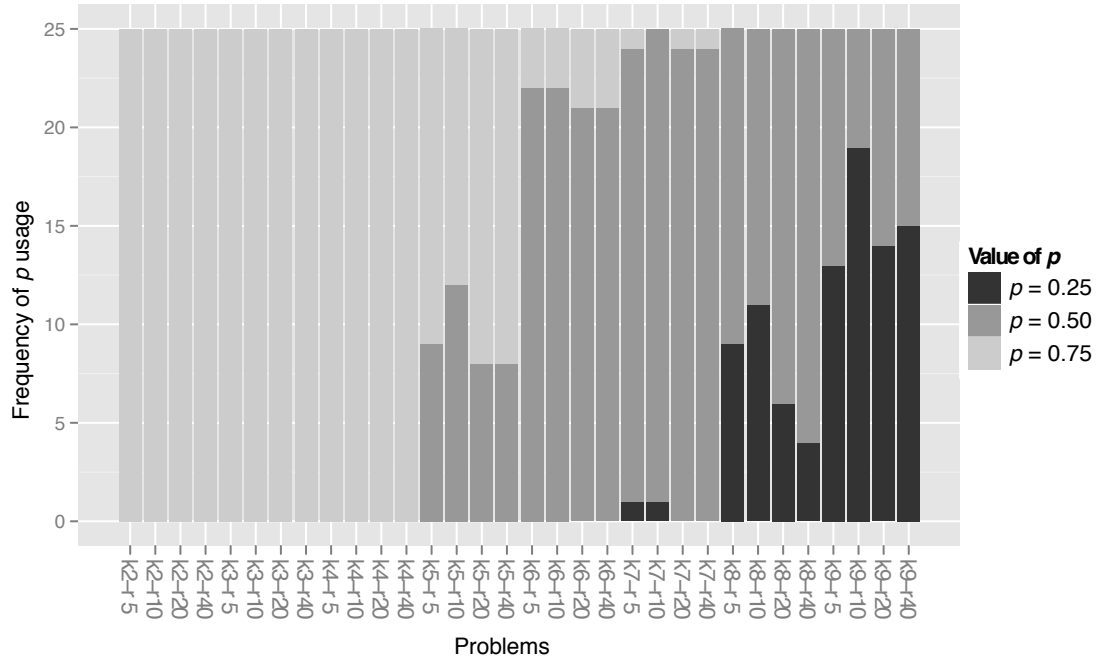
As we already explained in previous sections, our hyper-parameter control method involves an additional computational effort before the learning process. This extra effort of the heuristic includes the evaluation of the randomly initialised individuals used to find the adequate representatives plus the number of evaluations needed to prune the representatives.

Figure 7 shows the additional effort incurred by our approach for problems with no noise using a $minAcc = 1.0$. The effort is shown in terms of the number of rule evaluations (matching the rule against the complete training set and computing its accuracy). Execution time is not shown as it is proportional to the number of evaluations. In this figure, we can observe that while the k increases it becomes more expensive to run the hyper-parameter setting approach and more iterations are needed to find representatives.

Moreover, it is also noticeable a spiking behaviour in the additional effort. This behaviour is clarified by Figure 7b, which shows the frequency in which each value of p is selected. As it



(a) Number of evaluations



(b) Frequency of p usage on the 25 experiments run for each k-DNF variant

Figure 7: Number of rule evaluations and frequency of selection of a specific p value depending on problem characteristics (k, r) in our k-DNF experiments.

was explained before, our approach also adapts the p value to increase the probability of finding representatives when this task becomes very difficult. As we can see, the different stages in the behaviour of the effort observed in Figure 7 correspond to the transition stages between different values of p. When the system uses a smaller p the additional effort to find representatives becomes smaller. A further evaluation of the computational effort of the heuristic is available at [67].

6.2 Evaluation on a Real-world Problem

In order to test the performance of our method over real-world domains, we selected a binary protein structure prediction problem already used in [52]. Specifically, the problem being addressed is called *contact number prediction*. In this problem, for each amino-acid of a protein's sequence, the goal is to predict the number of other amino-acids that in the folded protein state are located at a distance less than a threshold d. In this case the contact number is binarised to (high/low) states to convert the problem into a binary classification dataset. The contact number state of an amino-acid is predicted from information about itself and its immediate ± 4 neighbours in the protein sequence. The information about each amino-acid (which roughly captures the hydrophobicity physico-chemical property of the amino-acid) is represented by a binary variable generated with the method presented in [52]. Hence, each instance in the dataset is represented by 9 binary attributes. This dataset has a total of 257,560 instances.

It is worth mentioning that this problem has a very high noise ratio, and there are no possible rules that have 100% accuracy. Therefore, in order to test the heuristic we have to relax the minimum accuracy required for the classifiers to be representatives to 0.7. This is the maximum value for *minAcc* for which our heuristic could actually identify representatives. The rest of the heuristic is applied to this dataset exactly as in the k-DNF datasets.

In this section we are going to analyse the results obtained with our approach by comparing them with an exhaustive experimentation to determine the adequate coverage breakpoint. In the exhaustive search we used coverage breakpoint values ranging from 2^{-2} to 2^{-9} (as this is the maximum possible since the problem has 9 attributes), and different values of p (0.75, 0.5 and 0.25). Since we are dealing with a real problem without a known structure, we don't know ahead of time what is the appropriate k for the dataset. Hence, the verification is going to be experimental by running the whole BioHEL algorithm, and determine the success of the heuristic based on the obtained test accuracy. The hyper-parameters for the BioHEL system are the ones uses in [27] except only for three hyper-parameters shown in Table 4. The "Initial MDL TL ratio" is used in the heuristic proposed in [59] to automatically tune the W hyper-parameter of BioHEL's fitness formula (equation 1). This hyper-parameter defines the expected contribution that the TL part of the fitness formula should have in good rules. The "number of windows in ILAS" is used within the Incremental Learning with Alternative Strata (ILAS) scheme employed by BioHEL to speed up its fitness evaluations. In ILAS the training set is divided into a certain number of strata, called *windows*. Each GA iteration uses a different window for its fitness computations using a round-robin policy.

For the analysis of the results, we will first apply a Wilcoxon pairwise test [68] to determine if there are significant differences between using different coverage breakpoints in this problem and which is the most adequate hyper-parameter value to solve it. Having found the adequate coverage breakpoint, we will then compare it with the results obtained by our approach. The results are going to be analysed in terms of accuracy, execution time and convergence time of both methodologies.

Regarding the dataset, this problem was separated in 10 training and test sets for ten-fold cross-validation. Each one of these problem instances was run with 5 different seeds, so the results are the average of 50 runs.

Table 5 shows the results in terms of accuracy, average time per run and the total time of the experiments when applying different coverage breakpoints of the type $cov = 2^{-k}$ where $k = \{2..9\}$. In this table, we can see that the coverage breakpoint that obtains the best results in terms of accuracy is 2^{-9}. Also, for this problem there are no differences between using different values of p in terms of accuracy or execution time. Moreover, in Table 6 we can see that the coverage breakpoint 2^{-9} is not significantly different than applying other values such as 2^{-7} or 2^{-8}. However, the maximum average accuracy is obtained with the smaller coverage breakpoint.

Table 4: Hyper-parameters for the BioHEL system and the heuristic to determine the problem structure.

BioHEL hyper-parameter	Value
GA Iterations	50
Initial MDL TL ratio	0.025
Number of windows in ILAS	20
Heuristic hyper-parameters	**Value**
Sample size - N	1000
Number of representatives needed - R	20

Table 5: Results over the binary PSP problem using fixed coverage breakpoints (where $cov(k) = 2^{-k}$) and different p values.

	cov (k)	Test Accuracy (%)	Average time (s)	Total time (s)
$p = 0.75$	2	70.15±0.96	244.67±120.89	12233.71
	3	70.27±0.84	427.00±207.95	21350.19
	4	71.46±0.56	378.64±207.36	18932.08
	5	71.79±0.47	386.40±147.90	19319.86
	6	72.12±0.45	640.26±238.88	32013.22
	7	72.32±0.52	874.15±322.80	43707.67
	8	72.40±0.47	1469.14±496.96	73457.19
	9	72.45±0.47	2526.63±723.67	126331.39
Total time				(≈96.48 h) 347345.31
$p = 0.5$	2	70.15±0.96	247.72±101.81	12385.91
	3	70.27±0.84	342.26±180.62	17112.77
	4	71.47±0.56	390.89±208.98	19544.62
	5	71.79±0.47	398.20±180.34	19909.83
	6	72.12±0.45	639.17±240.29	31958.73
	7	72.31±0.53	896.59±287.78	44829.29
	8	72.40±0.48	1601.17±445.91	80058.71
	9	72.45±0.47	2244.94±650.95	112246.99
Total time				(≈93.90 h) 338046.85
$p = 0.25$	2	70.46±0.84	239.53±91.12	11976.29
	3	70.27±0.84	362.16±191.81	18107.84
	4	71.46±0.56	332.23±183.90	16611.51
	5	71.80±0.48	409.98±167.93	20498.96
	6	72.12±0.45	622.01±230.56	31100.48
	7	72.31±0.53	846.66±272.08	42333.25
	8	72.40±0.48	1636.27±482.24	81813.37
	9	72.45±0.47	2338.56±586.86	116927.77
Total time				(≈94.26 h) 339369.47

Table 6: P-values of the Wilcoxon pairwise test to determine significant differences between the usage of different coverage breakpoints of the type 2^{-k} in the binary PSP problem. The cells in bold indicate the cases where there are significant differences.

P-values of the wilcoxon pairwise test							
k	2	3	4	5	6	7	8
3	0.52092	-	-	-	-	-	-
4	**2.9e-07**	**1.2e-08**	-	-	-	-	-
5	**1.5e-11**	**2.9e-11**	**0.01510**	-	-	-	-
6	**4.9e-15**	**4.9e-15**	**2.5e-06**	**0.00265**	-	-	-
7	**< 2e-16**	**< 2e-16**	**1.0e-08**	**0.00016**	0.21071	-	-
8	**< 2e-16**	**< 2e-16**	**6.1e-10**	**6.1e-06**	0.05857	0.57281	-
9	**< 2e-16**	**< 2e-16**	**7.5e-11**	**2.8e-06**	**0.01079**	0.21071	0.57281

Table 7: Performance of the heuristic over the binary PSP problem.

Min Acc	k	p	Test acc	Ave. Exec Time	Total time
0.7	8.98±0.14	0.28±0.08	72.45±0.47	2913.93±1356.41	145696.66
0.6	5.62±1.12	0.73±0.06	71.94±0.74	902.29±382.38	45114.63
Total time					(≈80.94 h) 291393.32

In Table 7 we can observe the results obtained with our approach. Using a minimum accuracy of 0.7 the system determined that the k of the problem is equal 9 in the majority of the cases and applying the corresponding coverage breakpoint we obtain an accuracy equal to our best results in preliminary experimentation. We also tested a more relaxed *minAcc* value without obtaining good results. Based on this results we can conclude that the heuristic, when using the appropriate noise ratio, is able to categorise this real problem correctly and determine the appropriate coverage breakpoint automatically.

Regarding the computational time, the total amount of CPU time invested in performing the preliminary experiments with the different coverage breakpoints and different values of p is equal to ≈285 hours. On the other hand, the amount of time invested in running the experiments automatically setting the coverage breakpoint, using different *minAcc* values and including the whole learning process was ≈81 hours. This constitutes 28% of the time invested in the preliminary experiments, which means a reduction of 71% in the total experimental time.

Moreover, our method also adapts the value p, the probability of setting the bits to 1 in the GABIL representation. Figure 8 reports the average accuracy of the best rule along the 50 iterations of the GA with different p values. So far the use of adapting this hyper-parameter was just to find representatives during the hyper-parameter control stage, but as this figure shows, using $p = 0.25$ makes the system find good classifiers quicker (although all three values of p eventually manage to learn the correct rules). This is because in this particular problem using a small value of p increases the odds of finding representatives. The spiking behaviour in the figure is a normal phenomena due to the usage of ILAS windowing scheme [59], and it is not related to the approach presented in this paper.

In this sense, we can say that the heuristic reduces the computational time needed to set up the algorithm properly, which for large problems can constitute a considerable amount of CPU

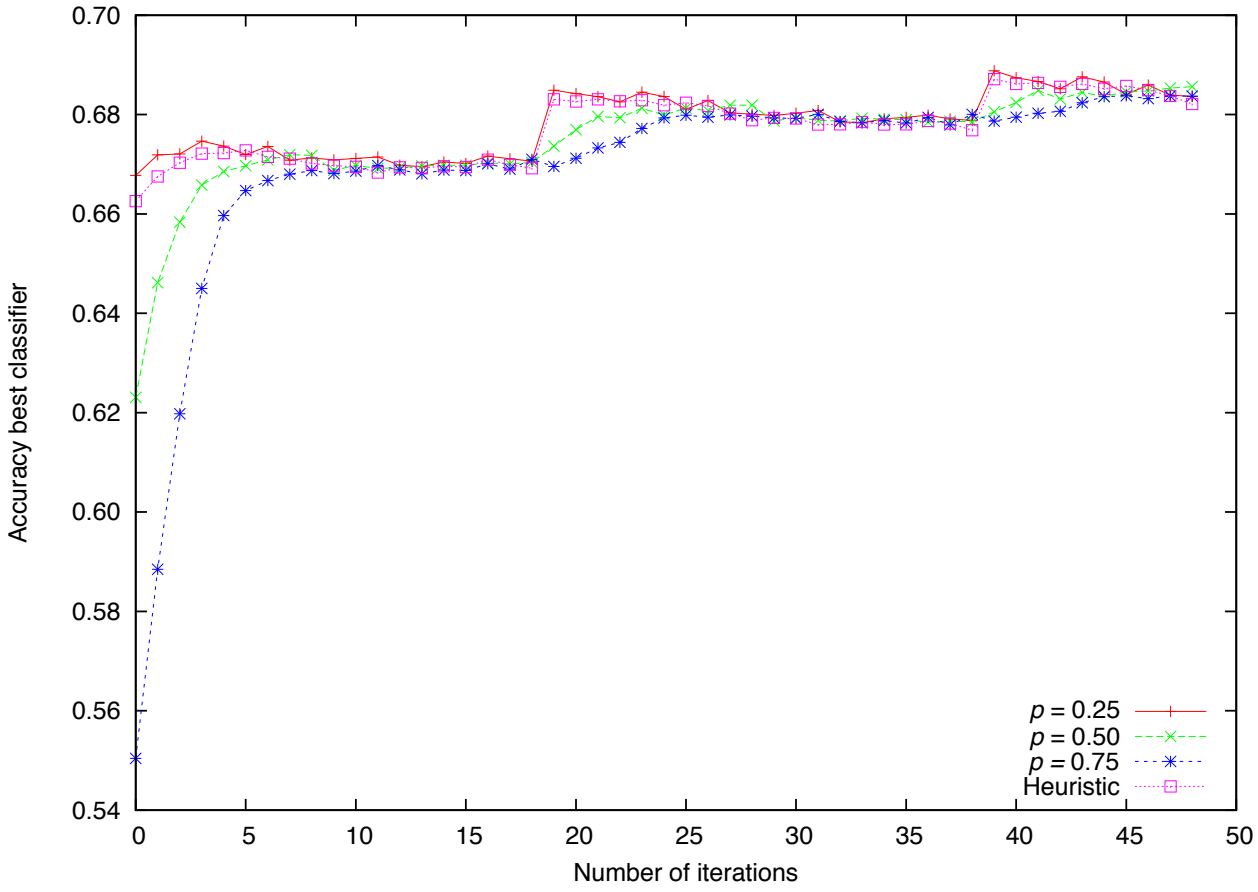


Figure 8: Average accuracy of the best classifier during the 50 iterations of the GA using different values of p.

time. Moreover, it adapts other hyper-parameters of the system, such as the p value, which helps finding good rules quicker within the genetic algorithm.

7 Discussion and further work

The initial objective of this paper was to design an automatic procedure to learn the structure of the problem and set the coverage breakpoint hyper-parameter in the BioHEL learning system for (a broad class of) binary classification domains. Our proposed approach does this by using simple models that correlate the behaviour of the system and characteristics of the data to the problem structure. Our thorough experiments show that our procedure is able to successfully estimate the problem's structure (k and r) in binary problems with noise (using many variants of the $k - KDF$ family of boolean functions) and in a real-world problem of protein structure prediction.

Using this method we are able to set up the coverage breakpoint hyper-parameter in BioHEL, facilitating in this way the learning process, reducing the computational time of preliminary experiments and making the system easier to use to an end-user. The final validation stage using a challenging protein structure prediction problem showed that the heuristics works for large real-world problems as well, managing to reduce the total experimental time by 71%. Moreover, our approach also adapts other hyper-parameters like p, the probability of settings bits to 1 in the GABIL representation, which can help finding good rules faster within the GA. And while the objective of this heuristic is to automatically tune crucial hyper-parameters of BioHEL, we are aware that we are introducing several new hyper-parameters. In our ex-

perimentation we show that most of these can remain at fixed values and still obtain good performance. The only crucial hyper-parameter to set up is *minAcc*, that specifies that minimum accuracy that the sample of initial rules should have to become representatives. As we argue in the paper, our view is that this hyper-parameter is much more intuitive to set up as it relates to the uncertainty of the dataset rather than being specific to the machine learning algorithm.

While the heuristic is designed with a specific system in mind, BioHEL, we believe that these design principles can be adapted to other machine learning algorithms to develop equivalent heuristics. First, the methodologies presented in this paper to find the structure of the problem can be extended to other systems by using models developed particularly for these systems based on the characteristics of the problem. Knowing in advance the characteristics of the problem at the beginning of the learning process can be advantageous to guide the search and also set hyper-parameters within the system. For instance, in the GAssist system [59] an adapted heuristic could be used to set the minimal rule set size penalty and the minimal number of rules for the rule deletion operator. For XCS, estimations of k could be used to automatically set up some of its hyper-parameters (e.g. mutation rate, population size), using the models proposed in [22]. Moreover, an adaptation of this heuristic could be used to seed the initial population, as suggested in [64].

The approach we decided to take in this paper is quite different from the *AutoML* approaches which broadly speaking (as explained in the related work section), apply a wrappers on top of the machine learning algorithm. These approaches have shown to be effective across many different domains but all they give you is the set of optimal hyper-parameters (or, in cases like TPOT, also the features selected) to maximise predictive performance. Our approach, while requiring a deep knowledge of the knowledge representation used within BioHEL, is able to estimate knowledge about the structure of the problem being studied, and at the same time as making the process of algorithm tuning easier for the users, is able to provide them with insight about the data.

Moreover, there are several potential lines to extend this work. Firstly, we would like to extend this approach to problems with χ-ary discrete[2] and continuous attributes. In the case of χ-ary attributes, the coverage would not only depend on k but would also depend in the number of values activated in an attribute (number of 1s in the GABIL representation), while for continuous domains coverage depends also on the size of the intervals, and would require a substantial (and challenging) rewrite of all the probabilistic models. If we can apply this heuristic to a more broad class of datasets then we can revisit our observations that most of the heuristic's parameters can remain fixed. Moreover, it would be interesting to investigate policies of selecting the k for the cases in which there is no single cell where the three metrics intersect. In these cases probably it is better to select a k based on the average of the cells that obtained the highest score. In relation to the tuning of the *minAcc* hyper-parameter of our heuristic, there are recent approaches to estimate the uncertainty of dataset labels [69], while it would also be interesting to revisit the use of complexity measures for classification datasets [49] to reliably estimate appropriate values for such hyper-parameter. As our experiments show, our heuristic struggles in scenarios with high rule overlap because k is underestimated as we are not able to identify good representatives. To this aim it would be interesting to explore how we could use *absumption* mechanisms [70] and other types of local search operators [65] to generate better representatives while keeping the computational effort of the heuristic under control.

Finally, a more extensive analysis of the additional effort required by this heuristic should be carried out, focusing on how the hyper-parameter R (target number of representatives to be generated) and the space sampling can reduce or regulate the extra computational effort,

[2]Discrete domains where each attribute has more than two possible values.

and estimating what is the worse-case effort that the heuristic would require. In relation to the adaptation of these approaches to other machine learning algorithms, we have pointed above several potential scenarios where this adaptation could take place. In such case this heuristic probably should be rewritten to become much more modular, and be able to separate the method/representation-specific components from those that could be reused across ML algorithms.

Acknowledgements

We are very grateful for the detailed feedback provided by the anonymous reviewers which has made this paper better. We acknowledge the support of the UK Engineering and Physical Sciences Research Council (EPSRC) under grants EP/H016597/1, EP/M020576/1 and EP/N031962/1.

References

[1] R. S. Olson, R. J. Urbanowicz, P. C. Andrews, N. A. Lavender, L. C. Kidd, and J. H. Moore, "Automating biomedical data science through tree-based pipeline optimization," in *Applications of Evolutionary Computation*, G. Squillero and P. Burelli, Eds. Cham: Springer International Publishing, 2016, pp. 123–137.

[2] F. Mohr, M. Wever, and E. Hüllermeier, "Ml-plan: Automated machine learning via hierarchical planning," *Machine Learning*, vol. 107, no. 8-10, pp. 1495–1515, 2018. [Online]. Available: https://doi.org/10.1007/s10994-018-5735-z

[3] M. López-Ibáñez, J. Dubois-Lacoste, L. Pérez Cáceres, M. Birattari, and T. Stützle, "The irace package: Iterated racing for automatic algorithm configuration," *Operations Research Perspectives*, vol. 3, pp. 43–58, 2016. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S2214716015300270

[4] M. Feurer, A. Klein, K. Eggensperger, J. Springenberg, M. Blum, and F. Hutter, "Efficient and robust automated machine learning," in *Advances in Neural Information Processing Systems 28*, C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, Eds. Curran Associates, Inc., 2015, pp. 2962–2970. [Online]. Available: http://papers.nips.cc/paper/5872-efficient-and-robust-automated-machine-learning.pdf

[5] A. Fernández, S. García, J. Luengo, E. Bernadó-Mansilla, and F. Herrera, "Genetics-based machine learning for rule induction: State of the art, taxonomy, and comparative study," *IEEE Transactions on Evolutionary Computation*, vol. 14, no. 6, pp. 913–941, Dec. 2010.

[6] A. Orriols-Puig, J. Casillas, and E. Bernadó-Mansilla, "Genetic-based machine learning systems are competitive for pattern recognition," *Evolutionary Intelligence*, vol. 1, pp. 209–232, 2008. [Online]. Available: http://dx.doi.org/10.1007/s12065-008-0013-9

[7] A. L. Swan, K. L. Hillier, J. R. Smith, D. Allaway, S. Liddell, J. Bacardit, and A. Mobasheri, "Analysis of mass spectrometry data from the secretome of an explant model of articular cartilage exposed to pro-inflammatory and anti-inflammatory stimuli using machine learning," *BMC Musculoskeletal Disorders*, vol. 14, no. 1, p. 349, 2013.

[8] A. Swan, D. Stekel, C. Hodgman, D. Allaway, M. Alqahtani, A. Mobasheri, and J. Bacardit, "A machine learning heuristic to identify biologically relevant and minimal biomarker panels from omics data," *BMC Genomics*, vol. 16, no. Suppl 1, p. S2, Jan. 2015.

[9] M. Martínez-Ballesteros, J. Bacardit, A. Troncoso, and J. C. Riquelme, "Enhancing the scalability of a genetic algorithm to discover quantitative association rules in large-scale datasets," *Integr. Comput.-Aided Eng.*, vol. 22, no. 1, pp. 21–39, Jan. 2015. [Online]. Available: https://doi.org/10.3233/ICA-140479

[10] S. Baron, N. Lazzarini, and J. Bacardit, "Characterising the influence of rule-based knowledge representations in biological knowledge extraction from transcriptomics data," in *Applications of Evolutionary Computation*, G. Squillero and K. Sim, Eds. Cham: Springer International Publishing, 2017, pp. 125–141.

[11] Y. Bi, B. Xue, and M. Zhang, "An automated ensemble learning framework using genetic programming for image classification," in *Proceedings of the Genetic and Evolutionary Computation Conference*, ser. GECCO 2019. New York, NY, USA: Association for Computing Machinery, 2019, pp. 365–373. [Online]. Available: https://doi.org/10.1145/3321707.3321750

[12] J. Liang, E. Meyerson, and R. Miikkulainen, "Evolutionary architecture search for deep multitask networks," in *Proceedings of the Genetic and Evolutionary Computation Conference*, ser. GECCO 2018. New York, NY, USA: Association for Computing Machinery, 2018, pp. 466–473. [Online]. Available: https://doi.org/10.1145/3205455.3205489

[13] A. Sohn, R. S. Olson, and J. H. Moore, "Toward the automated analysis of complex diseases in genome-wide association studies using genetic programming," in *Proceedings of the Genetic and Evolutionary Computation Conference*, ser. GECCO 2017. New York, NY, USA: Association for Computing Machinery, 2017, pp. 489–496. [Online]. Available: https://doi.org/10.1145/3071178.3071212

[14] A. E. Eiben, Z. Michalewicz, M. Schoenauer, and J. E. Smith, *Parameter Control in Evolutionary Algorithms*, ser. Studies in Computational Intelligence. Springer Berlin Heidelberg, 2007, vol. 54, ch. chapter 2, pp. 19–46. [Online]. Available: http://www.springerlink.com/index/10.1007/978-3-540-69432-8_2

[15] A. E. Eiben, M. Horvath, W. Kowalczyk, and M. C. Schut, *Reinforcement Learning for Online Control of Evolutionary Algorithms*, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2007, vol. 4335, ch. chapter 10, pp. 151–160. [Online]. Available: http://www.springerlink.com/index/10.1007/978-3-540-69868-5_10

[16] J. Smith and T. C. Fogarty, *Adaptively parameterised evolutionary systems: Self adaptive recombination and mutation in a genetic algorithm*, ser. Lecture Notes in Computer Science. Springer-Verlag, 1996, vol. 1141, ch. chapter 45, pp. 441–450. [Online]. Available: http://www.springerlink.com/index/10.1007/3-540-61723-X_1008

[17] J. Smith and T. Fogarty, "Self adaptation of mutation rates in a steady state genetic algorithm," in *Proceedings of the IEEE International Conference on Evolutionary Computation 1996*, May 1996, pp. 318–323.

[18] N. Krasnogor and J. Smith, "A memetic algorithm with self-adaptive local search: TSP as a case study," in *GECCO*, L. D. Whitley, D. E. Goldberg, E. Cantú-Paz, L. Spector, I. C. Parmee, and H.-G. Beyer, Eds. Morgan Kaufmann, 2000, pp. 987–994.

[19] ——, "Emergence of profitable search strategies based on a simple inheritance mechanism," in *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO2001)*. San Francisco, CA: Morgan Kaufmann Publishers, 2001, pp. 432–439. [Online]. Available: http://www.cs.nott.ac.uk/~nxk/PAPERS/ga140.pdf

[20] N. Krasnogor and S. Gustafson, "A study on the use of "self-generation" in memetic algorithms," *Natural Computing*, vol. 3, no. 1, pp. 53–76, Mar. 2004. [Online]. Available: http://dx.doi.org/10.1023/B:NACO.0000023419.83147.67

[21] N. Krasnogor, "Self generating metaheuristics in bioinformatics: The proteins structure comparison case," *Genetic Programming and Evolvable Machines*, vol. 5, no. 2, pp. 181–201, Jun. 2004. [Online]. Available: http://dx.doi.org/10.1023/B:GENP.0000023687.41210.d7

[22] M. V. Butz, *Rule-Based Evolutionary Online Learning Systems: A Principled Approach to LCS Analysis and Design*, ser. Studies in Fuzziness and Soft Computing. Springer, 2006, vol. 109.

[23] A. Orriols-Puig, X. Llorà, and D. E. Goldberg, "How XCS deals with rarities in domains with continuous attributes," in *GECCO'10*, 2010, pp. 1023–1030.

[24] P. O. Stalph, X. Llorà, D. E. Goldberg, and M. V. Butz, "Resource management and scalability of the XCSF learning classifier system," *Theoretical Computer Science*, vol. 425, pp. 126–141, March 2012. [Online]. Available: http://dx.doi.org/10.1016/j.tcs.2010.07.007

[25] M. Nakata, W. Browne, T. Hamagami, and K. Takadama, "Theoretical XCS parameter settings of learning accurate classifiers," in *Proceedings of the Genetic and Evolutionary Computation Conference*, ser. GECCO 2017. New York, NY, USA: Association for Computing Machinery, 2017, pp. 473–480. [Online]. Available: https://doi.org/10.1145/3071178.3071200

[26] M. Nakata, W. Browne, and T. Hamagami, "Theoretical adaptation of multiple rule-generation in XCS," in *Proceedings of the Genetic and Evolutionary Computation Conference*, ser. GECCO 2018. New York, NY, USA: Association for Computing Machinery, 2018, pp. 482–489. [Online]. Available: https://doi.org/10.1145/3205455.3205465

[27] J. Bacardit, E. K. Burke, and N. Krasnogor, "Improving the scalability of rule-based evolutionary learning," *Memetic Computing*, vol. 1, no. 1, pp. 55–67, Mar. 2009. [Online]. Available: http://dx.doi.org/10.1007/s12293-008-0005-4

[28] J. Bacardit, P. Widera, A. Márquez-Chamorro, F. Divina, J. S. Aguilar-Ruiz, and N. Krasnogor, "Contact map prediction using a large-scale ensemble of rule sets and the fusion of multiple predicted structural features," *Bioinformatics*, vol. 28, no. 19, pp. 2441–2448, Oct 2012. [Online]. Available: http://bioinformatics.oxfordjournals.org/content/28/19/2441.abstract

[29] E. Glaab, J. Bacardit, J. M. Garibaldi, and N. Krasnogor, "Using rule-based machine learning for candidate disease gene prioritization and sample classification of cancer gene expression data," *PLoS ONE*, vol. 7, no. 7, p. e39932, Jul 2012. [Online]. Available: http://dx.doi.org/10.1371\%2Fjournal.pone.0039932

[30] M. A. Franco, N. Krasnogor, and J. Bacardit, "Analysing BioHEL using challenging boolean functions," *Evolutionary Intelligence*, vol. 5, pp. 87–102, Jun 2012, 10.1007/s12065-012-0080-9. [Online]. Available: http://dx.doi.org/10.1007/s12065-012-0080-9

[31] ——, "Modelling the initialisation stage of the ALKR representation for discrete domains and GABIL encoding," in *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation*, ser. GECCO '11. New York, NY, USA: ACM Press, 2011, pp. 1291–1298. [Online]. Available: http://doi.acm.org/10.1145/2001576.2001750

[32] I. Rechenberg, *Evolutionsstrategie: Optimierung Technischer Systeme Nach Prinzipien der Biologischen Evolution*. Frommann-Holzboog, 1973.

[33] H. Schwefel, "Evolutionsstrategie und numerische optimierung," Dr.-Ing. Thesis, Technical University of Berlin, Department of Process Engineering, 1975.

[34] W. M. Spears, "Adapting crossover in evolutionary algorithms," *Proceeding of the 4th annual conference on Evolutionary Programming*, pp. 367—384, 1995. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.49.9322

[35] L. Davis, "Adapting operator probabilities in genetic algorithms," in *Proceedings of the third International Conference on Genetic Algorithms*. George Mason University, United States: Morgan Kaufmann Publishers Inc., 1989, pp. 61–69. [Online]. Available: http://portal.acm.org/citation.cfm?id=93146

[36] A. G. Carvalho and A. F. Araujo, "Improving NSGA-II with an adaptive mutation operator," in *Proceedings of the 11th Annual Conference Companion on Genetic and Evolutionary Computation Conference: Late Breaking Papers*. Montreal, Québec, Canada: ACM, 2009, pp. 2697–2700. [Online]. Available: http://portal.acm.org/citation.cfm?id=1570387

[37] L. DaCosta, Álvaro Fialho, M. Schoenauer, and M. Sebag, "Adaptive operator selection with dynamic multi-armed bandits," in *Proceedings of the 10th Annual Conference on Genetic and Evolutionary Computation*. Atlanta, GA, USA: ACM Press, 2008, pp. 913–920. [Online]. Available: http://portal.acm.org/citation.cfm?id=1389272

[38] S. Müller, N. N. Schraudolph, and P. D. Koumoutsakos, "Step size adaptation in evolution strategies using reinforcement learning," in *Proceedings of the 2002 IEEE Congress on Evolutionary Computation*. IEEE Press, 2002, pp. 151–156.

[39] Y. Sakurai, K. Takada, T. Kawabe, and S. Tsuruta, "A method to control parameters of evolutionary algorithms by using reinforcement learning," in *Proceedings of the Sixth International Conference on Signal-Image Technology and Internet Based Systems*. IEEE, Dec 2010, pp. 74–79. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5714532

[40] S. W. Wilson, "ZCS: a zeroth level classifier system," *Evol. Comput.*, vol. 2, no. 1, pp. 1–18, Mar. 1994. [Online]. Available: http://dx.doi.org/10.1162/evco.1994.2.1.1

[41] L. Bull and J. Hurst, *Self-Adaptive Mutation in ZCS Controllers*, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2000, vol. 1803, ch. chapter 33, pp. 342–349. [Online]. Available: http://www.springerlink.com/index/10.1007/3-540-45561-2_33

[42] J. Hurst and L. Bull, *A Self-Adaptive Classifier System*, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2001, vol. 1996, ch. chapter 6, pp. 70–79. [Online]. Available: http://www.springerlink.com/index/10.1007/3-540-44640-0_6

[43] S. W. Wilson, "Classifier fitness based on accuracy," *Evol. Comput.*, vol. 3, no. 2, pp. 149–175, Jun. 1995. [Online]. Available: http://dx.doi.org/10.1162/evco.1995.3.2.149

[44] J. Hurst and L. Bull, "A self-adaptive XCS," in *Advances in Learning Classifier Systems*, ser. Lecture Notes in Computer Science, P. Lanzi, W. Stolzmann, and S. Wilson, Eds. Springer Berlin / Heidelberg, 2002, vol. 2321, pp. 333–360, 10.1007/3-540-48104-4_5. [Online]. Available: http://dx.doi.org/10.1007/3-540-48104-4_5

[45] S. W. Wilson, "Classifiers that approximate functions," *Natural Comput.*, vol. 1, no. 2-3, pp. 211–234, 2002. [Online]. Available: http://portal.acm.org/citation.cfm?id=599708

[46] M. V. Butz, P. O. Stalph, and P. L. Lanzi, "Self-adaptive mutation in XCSF," in *Proceedings of the 10th Annual Conference on Genetic and Evolutionary Computation*. Atlanta, GA, USA: ACM Press, 2008, pp. 1365–1372. [Online]. Available: http://portal.acm.org/citation.cfm?id=1389095.1389361

[47] M. Ghallab, D. Nau, and P. Traverso, *Automated Planning: Theory and Practice*. Elsevier, 2004.

[48] O. Maron and A. W. Moore, "The racing algorithm: Model selection for lazy learners," *Artificial Intelligence Review*, vol. 11, no. 1-5, pp. 193–225, 1997.

[49] T. K. Ho and M. Basu, "Complexity measures of supervised classification problems," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24, no. 3, pp. 289–300, Mar. 2002.

[50] E. Bernado-Mansilla and T. K. Ho, "Domain of competence of XCS classifier system in complexity measurement space," *IEEE Transactions on Evolutionary Computation*, vol. 9, no. 1, pp. 82–104, Feb. 2005.

[51] J. Luengo, A. Fernández, S. García, and F. Herrera, "Addressing data complexity for imbalanced data sets: Analysis of SMOTE-based oversampling and evolutionary undersampling," *Soft Computing*, vol. 15, no. 10, pp. 1909–1936, 2011.

[52] J. Bacardit, M. Stout, J. D. Hirst, A. Valencia, R. Smith, and N. Krasnogor, "Automated alphabet reduction for protein datasets," *BMC Bioinformatics*, vol. 10, no. 1, p. 6, 2009. [Online]. Available: http://www.biomedcentral.com/1471-2105/10/6

[53] G. W. Bassel, E. Glaab, J. Marquez, M. J. Holdsworth, and J. Bacardit, "Functional network construction in arabidopsis using rule-based machine learning on large-scale data sets," *The Plant Cell Online*, vol. 23, no. 9, pp. 3101–3116, Sep. 2011.

[54] M. Stout, J. Bacardit, J. D. Hirst, and N. Krasnogor, "Prediction of recursive convex hull class assignments for protein residues," *Bioinformatics*, vol. 24, no. 7, pp. 916–923, Apr. 2008. [Online]. Available: http://bioinformatics.oxfordjournals.org/cgi/content/abstract/24/7/916

[55] G. Venturini, "SIA: a supervised inductive algorithm with genetic search for learning attributes based concepts," in *Machine Learning: ECML-93 - Proceedings of the European Conference on Machine Learning*, P. B. Brazdil, Ed. Springer-Verlag, 1993, pp. 280–296.

[56] J. Bacardit and N. 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*. New York, NY, USA: ACM Press, 2009, pp. 1155–1162.

[57] S. W. Wilson, "Mining oblique data with XCS," in *Advances in Learning Classifier Systems*, ser. Lecture Notes in Computer Science, P. Luca Lanzi, W. Stolzmann, and S. Wilson, Eds. Springer Berlin / Heidelberg, 2001, vol. 1996, pp. 283–290. [Online]. Available: http://dx.doi.org/10.1007/3-540-44640-0_11

[58] K. D. Jong and W. M. Spears, "Learning concept classification rules using genetic algorithms," in *Proceedings of the 12th International Joint Conference on Artificial Intelligence - Volume 2*. Sydney, New South Wales, Australia: Morgan Kaufmann Publishers Inc., 1991, pp. 651–656. [Online]. Available: http://portal.acm.org/citation.cfm?id=1631559

[59] J. Bacardit, "Pittsburgh Genetics-Based machine learning in the data mining era: Representations, generalization, and run-time," PhD thesis, Ramon Llull University, Barcelona, Spain, 2004.

[60] J. Rissanen, "Modeling by shortest data description," *Automatica*, vol. vol. 14, pp. 465–471, 1978.

[61] M. J. Kearns, *The Computational Complexity of Machine Learning*. MIT Press, Cambridge, Massachusetts, 1990.

[62] M. V. Butz and M. Pelikan, "Studying XCS/BOA learning in Boolean functions: structure encoding and random Boolean functions," in *GECCO '06: Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation*. New York, NY, USA: ACM Press, 2006, pp. 1449–456.

[63] A. Hernández-Aguirre, B. P. Buckles, and C. A. C. Coello, "On learning $kDNF_n^s$s boolean formulas," in *NASA/DoD Conference on Evolvable Hardware*, vol. 0. Los Alamitos, CA, USA: IEEE Computer Society, 2001, p. 0240.

[64] C. Ioannides, G. Barrett, and K. Eder, "XCS cannot learn all boolean functions," in *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation*, ser. GECCO '11. New York, NY, USA: ACM Press, 2011, pp. 1283–1290. [Online]. Available: http://doi.acm.org/10.1145/2001576.2001749

[65] M. A. Franco, N. Krasnogor, and J. Bacardit, "Post-processing operators for decision lists," in *Proceedings of the 14th International Conference on Genetic and Evolutionary Computation Conference*, ser. GECCO '12. New York, NY, USA: ACM Press, 2012, pp. 847–854. [Online]. Available: http://doi.acm.org/10.1145/2330163.2330281

[66] J. Demšar, "Statistical comparisons of classifiers over multiple data sets," *J. Mach. Learn. Res.*, vol. 7, pp. 1–30, 2006. [Online]. Available: http://portal.acm.org/citation.cfm?id=1248548

[67] M. A. Franco, "Principled design of evolutionary learning systems for large scale data mining," PhD thesis, Universitat of Nottingham, 2013.

[68] F. Wilcoxon, "Individual comparisons by ranking methods," *Biometrics*, vol. 1, no. 6, pp. 80–83, 1945. [Online]. Available: http://www.jstor.org/stable/3001968

[69] C. G. Northcutt, L. Jiang, and I. L. Chuang, "Confident learning: Estimating uncertainty in dataset labels," arXiv:1911.00068, 2019.

[70] Y. Liu, W. N. Browne, and B. Xue, "Absumption to complement subsumption in learning classifier systems," in *Proceedings of the Genetic and Evolutionary Computation Conference*, ser. GECCO 19. New York, NY, USA: Association for Computing Machinery, 2019, pp. 410–418. [Online]. Available: https://doi.org/10.1145/3321707.3321719