Measuring Similarities in Model Structure of Metaheuristic Rule Set Learners

David Pätzel[1(✉)], Richard Nordsieck[2], and Jörg Hähner[1]

[1] University of Augsburg, Augsburg, Germany
david.paetzel@uni-a.de
[2] XITASO GmbH IT & Software Solutions, Augsburg, Germany

Abstract. We present a way to measure similarity between sets of rules for regression tasks. This was identified to be an important but missing tool to investigate Metaheuristic Rule Set Learners (MRSLs), a class of algorithms that utilize metaheuristics such as Genetic Algorithms to solve learning tasks: The commonly-used predictive performance-based metrics such as mean absolute error do not capture most users' actual preferences when they choose these kinds of models since they typically aim for model interpretability (i. e. low number of rules, meaningful rule placement etc.) and not low error alone. Our similarity measure is based on a form of metaheuristic-agnostic edit distance. It is meant to be used—in conjunction with a certain class of benchmark problems—for analysing and improving an as-of-yet underresearched part of MRSL algorithms: The metaheuristic that optimizes the model's structure (i. e. the set of rule conditions). We discuss the measure's most important properties and demonstrate its applicability by performing experiments on the best-known MRSL, XCSF, comparing it with two non-metaheuristic Rule Set Learners, Decision Trees and Random Forests.

Keywords: Benchmarking · Metaheuristic Rule Learning · Model Similarity · Learning Classifier Systems · Rule Learning

1 Introduction

This paper presents a novel tool for evaluating *Metaheuristic Rule Set Learners* (MRSL) for *regression tasks*. MRSLs are a subclass of *Rule Set Learners* (RSLs) which are machine learning (ML) algorithms that learn *sets of rules*. While non-metaheuristic RSLs like the well-known C4.5 algorithm [34] typically use local heuristics to generate sets of rules, MRSLs use metaheuristics such as Genetic Algorithms (GAs) to perform some form of global search for well-performing rule sets. Examples for MRSL systems are Learning Classifier Systems such as XCS [41], Fuzzy Rule-based Systems [7] or Ant-Miner [26]. While there are MRSL approaches for unsupervised learning (e. g. [38]), reinforcement learning (e. g. [5]) and classification (e. g. [3,26]) as well, the present paper focusses on *regression tasks*. MRSL systems solving regression tasks include [2,17,42].

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2024
S. Smith et al. (Eds.): EvoApplications 2024, LNCS 14635, pp. 256–272, 2024.
https://doi.org/10.1007/978-3-031-56855-8_16

In the regression tasks that we are concerned with, the goal is to find an in some sense optimal model $\widehat{f} : \mathcal{X} \to \mathcal{Y}$ that maps inputs $x \in \mathcal{X} = \mathbb{R}^{\mathcal{D}_\mathcal{X}}$ to outputs $y \in \mathcal{Y} = \mathbb{R}$. At that, the only guidance given in order to find that optimal model is a *training set* consisting of N inputs $(x_n)_{n=1}^N \subsetneq \mathcal{X}$ and outputs $(y_n)_{n=1}^N \subsetneq \mathcal{Y}$. A common way to measure and improve model optimality is to compute predictive error measures such as the *mean absolute error* (MAE) or the *mean squared error*. While MRSLs are commonly evaluated using these kinds of metrics as well, Pätzel et al. [27] as well as Kovacs and Kerber [20,21] argue that during their development, MRSLs should actually be handled differently due to the fact that these algorithms not only optimize some parametric model's fixed set of *parameters* (like, for example, fitting neural network connection weights) but actually optimize the number, the conditions *and* the model parameters of a set of rules. This means that MRSLs actually perform both parameter optimization and *model structure* optimization (which in the neural network example translates to fitting connection weights *and* optimizing the network's architecture). This dual nature of MRSL algorithms entails that predictive error measures alone can typically not capture a user's actual preferences: They chose MRSL algorithms over other high-predictive-performance options such as Neural Networks for their increased interpretability and therefore also require the created model to have a low number of rules, meaningfully placed rules, little rule overlap and similar properties. That being said, the optimization target of supervised MRSL algorithms is far from clear and that may be one of the reasons why most MRSL research has focussed on improving these systems' predictive performance alone.

Pätzel et al. [27] present a concept to resolve that mismatch between MRSL research targets and MRSL user preferences. That concept is based on generating certain data-generating processes that serve as a new form of benchmark learning tasks for MRSL algorithms (this is summarized in Sect. 3). The advantage of these processes over other approaches is that they are of the same form as the models built by MRSL algorithms: Each process is a set of rules. The goal is to enable the following workflow for investigating MRSL algorithms:

1. Generate a random data-generating process (or, rather, many of them).
2. Generate training data using that process.
3. Apply an MRSL algorithm to the generated training data.
4. Compare the model (a set of rules) created by the MRSL algorithm with the set of rules of the original data-generating process.

At that, the very last step is what Pätzel et al.'s proposal is all about: The data-generating process being a set of rules allows to not only consider predictive performance but also *whether the MRSL algorithm was able to reconstruct the original data-generating process*. This enables directly measuring the progress made by the MRSL algorithm's *metaheuristic* since the model structures of the learnt model and of the data-generating process can be compared.

While Pätzel et al. [27] explained the overall concept of these principled benchmarks such as how data-generating processes can be generated, their paper fell short of proposing an actual way to compare model structures consisting of

sets of rules. This is where the present paper comes in: We identify and discuss the desired properties of dissimilarity measures that could be used for this task and then *present a novel dissimilarity measure between sets of rules* that fulfills them. In combination with above-mentioned benchmark tasks, the measure allows improving characteristics of MRSL algorithms other than predictive performance. Specifically, in the context of XAI, this allows to play to the strengths of MRSL (inherent explainability) and quantify the tradeoff between raw predictive performance and comprehensible model structures. We demonstrate our measure's applicability by applying it to the analysis of differences between several parametrizations of one of the best-known MRSLs, XCSF, and two non-metaheuristic RSLs, namely DTs and *Random Forests* (RFs).

2 Metaheuristic Rule Set Learners

In this section, we try to give a, due to space restrictions very rough, idea of the models built and assumptions made by *Metaheuristic Rule Set Learners* (MRSLs; for more details see [27]). This is necessary in order to be able to describe in Sect. 3 how the benchmark learning tasks look like and in Sect. 4 the dissimilarity measure between model structures.

As was already said above, this paper focusses on regression tasks (i. e. learning a mapping $\mathcal{X} \to \mathcal{Y}$ with $\mathcal{X} = \mathbb{R}^{\mathcal{D}_\mathcal{X}}$ and $\mathcal{Y} = \mathbb{R}$, $\mathcal{D}_\mathcal{X} \in \mathbb{N}$). Common and well-established MRSLs (e. g. XCSF [42]) as well as more recently developed systems (e. g. SupRB [17]) solve regression tasks by building discriminative models of the following form [27]:

$$\widehat{f}_\mathcal{M}(\theta, x) = \sum_{k=1}^{K} m(\psi_k; x)\, \gamma_k\, \widehat{f}_k(\theta_k; x) \tag{1}$$

At that,

- $m(\psi_k; \cdot) : \mathcal{X} \to \{0, 1\}$ is the *condition* or *matching function* of rule k (parametrized by ψ_k) which states for any $x \in \mathcal{X}$ whether rule k applies or not and correspondingly whether it will influence the overall prediction for that input (if $m(\psi_k; x) = 1$, we say $m(\psi_k; \cdot)$ *matches* data point x),
- γ_k is the *mixing weight* of rule k which allows to weigh rules against each other in areas of overlap,
- $\widehat{f}(\theta_k; \cdot) : \mathcal{X} \to \mathcal{Y}$ is the *local model* of rule k (parametrized by θ_k and fitted on the subset of the training data that $m(\psi_k; \cdot)$ matches) which gives the rule's output for any input $x \in \mathcal{X}$,
- the model's *parameters* form $\theta = \left((\gamma_k)_{k=1}^K, (\theta_k)_{k=1}^K\right)$,
- the model's *model structure* is $\mathcal{M} = \left(K, (\psi_k)_{k=1}^K\right)$.

Given a certain fixed model structure, a model's parameters' optimization is often straightforward: Each local model k's parameters θ_k only have to be optimized on a subset of the training data (i. e. the data points where the local

model's condition is fulfilled) and local model families are typically simple such as linear regression models or just constants [27]. While optimal mixing weights γ_k are often computationally expensive to obtain, there exist well-performing heuristics which are often-used in MRSL algorithms [9].

Other than model parameter optimization, optimization of the model structure is a difficult task and, correspondingly, most of the compute of (M)RSLs goes into doing so. While non-evolutionary RSLs such as *Decision Trees* (DTs) choose the model structure based on (often, local) heuristics, MRSLs use meta-heuristics and often some form of global search (e. g. Genetic Algorithms [41]); a recent overview of techniques used was given by Heider et al. [14]. How exactly model structure optimization is done strongly depends on the condition family used. For real-valued tasks such as the regression tasks considered in the present paper, a common choice are interval-based conditions where

$$m(\psi_k, x) = m((l_k, u_k), x) = \begin{cases} 1, & x \in [l_k, u_k) \\ 0, & \text{otherwise.} \end{cases} \tag{2}$$

Since intervals are easily comprehensible greater-/less-than statements, interval-based conditions often yield models with higher interpretability than more sophisticated condition families (e. g. ellipsoid-based matching [37]) and are thus often preferred [15]. At the same time, they are reasonably expressive and share a lot of similarity with other axis-parallel ways to subdivide the input space (e. g. the axis-parallel cuts made by DT algorithms); the latter being especially important if comparisons with such algorithms are being conducted—which we do in this paper. Overall, this led us to use interval-based conditions for the purposes of this paper as well and as a result, the $(m(\psi_k, \cdot))_{k=1}^K$ correspond to (and are sometimes called) a *set of ($\mathcal{D}_\mathcal{X}$-dimensional) intervals*.

3 Generating Benchmark Tasks

As was already said in the introduction, Pätzel et al.'s framework [27] is based on generating a set of benchmark learning tasks that have the same form as the common MRSL models introduced in Sect. 2 (cf. Eq. (1)). This means that each benchmark learning task is a data-generating process which corresponds to, for each input $x \in \mathcal{D}_\mathcal{X}$, a random variable of the following form:

$$Y = \sum_{k=1}^{K} m(\psi_k; x)\, \gamma_k \; (f_k(\theta_k; x) + \epsilon_k) \tag{3}$$

where the $\{\epsilon_k\}_{k=1}^K$ are normally distributed random variables corresponding to the respective local model's noise. These processes can be generated randomly by drawing all of the required parameters from suitable random distributions.

Compared to the original work [27,33], we introduced a minimum coverage parameter that allows us to ensure that the generated rules properly cover the input space. Since the exact way how the learning tasks are generated is not

relevant for the discussion of our dissimilarity measure, we refer to our code [31] for details on this. We further changed the local model family to constant models since they are less expensive to compute and the algorithms we use for our demonstration in Sect. 5 use constant local models as well.

4 Measuring Dissimilarity of Sets of Rules

This section describes our proposal for measuring dissimilarity between the model structures of models generated by MRSL algorithms. There are many existing ways (e. g. the Hausdorff distance [10] or intersection over union) to measure the dissimilarity between sets and most can be adapted to work on the sets corresponding to MRSL model structures (i. e. sets of rule conditions). However, due to the nature of MRSL model structures, the dissimilarity measure should fulfill certain properties which we found difficult or impossible to fulfill using the available dissimilarity measures. We start this section with a discussion of said properties and then introduce our dissimilarity measure.

4.1 Desired Properties

A dissimilarity measure $d(\cdot, \cdot)$ for MRSL model structures should have the following properties:

Property 1 (Symmetry). Since we may not only want to examine dissimilarities to the data-generating process model but also between two models generated by two different (or even the same) algorithm, we want the dissimilarity measure to be symmetric in order to avoid having to choose which of the two models should be used as a reference. Symmetry can be expressed formally as

$$d(\mathcal{M}_1, \mathcal{M}_2) = d(\mathcal{M}_2, \mathcal{M}_1) \tag{4}$$

Property 2 (Same training match sets[1] should yield minimal dissimilarity). While the training data and with it, the input space $\mathcal{X}$, is typically scaled (e. g. min-max normalized) before it is fed into an ML algorithm, we still would like the dissimilarity measure to be agnostic of input-space-based distances between interval bounds. The reason for this is that we do not want to punish cases where a metaheuristic has *bad luck*: Any predictive-performance-based part of a fitness function of a metaheuristic in an MRSL algorithms can only ever distinguish model structures if there is a change in at least one rule's training match set (See footnote 1). If we did not require this property, then a model structure $\mathcal{M}_1$ may be considered less or more similar to a reference model structure $\mathcal{M}_0$ than another model structure $\mathcal{M}_2$ despite $\mathcal{M}_1$ and $\mathcal{M}_2$ resulting in the exact same overall model with respect to the training data (i. e. despite differing ψ_k, all match sets are equal due to where in input space the training data points

[1] A *rule k's training match set* is the set of training data points that $m(\psi_k; \cdot)$ is fulfilled for, i. e. $\{x \in X \mid m(\psi_k; x) = 1\}$.

lie). Since most metaheuristics are non-deterministic, we conjecture that this occurring is not merely pathological. In short: If this property is fulfilled then two model structures that induce the same training data set for each local model have zero dissimilarity—even if the model structure parameters are actually different. Formally,[2]

$$(m_k(\psi_{1k}; X))_{k=1}^{K_1} = (m_k(\psi_{2k}; X))_{k=1}^{K_2} \quad \Leftrightarrow \quad d(\mathcal{M}_1, \mathcal{M}_2) = 0 \tag{5}$$

Property 3 (Different training match sets should yield non-minimal dissimilarity). This is somewhat of an inverse of Property 2. Whenever two model structures differ enough that the training data of at least one of the local models changes, the two model structures should not be considered maximally similar:

$$(m_k(\psi_{1k}; X))_{k=1}^{K_1} \neq (m_k(\psi_{2k}; X))_{k=1}^{K_2} \quad \Leftrightarrow \quad d(\mathcal{M}_1, \mathcal{M}_2) > 0 \tag{6}$$

Property 4 (Sensible behaviour even if conditions do not overlap). Even if two conditions (or sets of conditions) do *not* overlap, they can still be more or less similar to each other from both a metaheuristics point of view and also from a human user's perspective. For example, a single condition (corresponding to a set of rules of size one) should be seen as more similar to a set of conditions that lies closer to it than to a set that lies further away—even if it does not overlap with either of both sets. We therefore want the dissimilarity measure to be able to sensibly differentiate between pairs of conditions even if these conditions do not overlap. Since the concept of *sensibility* is rather vague, we deliberately do not try to formalize this property.

4.2 Dissimilarity Measure

Before we define our dissimilarity measure formally, we try to provide some intuition about it. The dissimilarity measure is ultimately meant to allow investigating the progress and behaviour of metaheuristics which optimize the model structures of MRSL models. As previously mentioned we consider MRSLs with interval-based conditions and hard binary matching (i. e. either a training data point is matched or it is not). For these MRSLs, a finite data supervised learning setting effectively induces a form of quantization of the training data signal when the model structure is changed: The model's output for the training data will only be different if at least one of the conditions is changed enough that at least one of the local models is fitted on a different subset of the training data. This gave us the idea of developing a measure similar to *edit distances* which are often used in discrete spaces (e. g. in graphs [12]): We compute an upper bound on the maximum number of training data prediction-changing edits required to transform rules into each other. This can be seen as an edit operator-agnostic

[2] We slightly abuse notation here and overload the matching function m to be able to pass the training data input $N \times \mathcal{D}_{\mathcal{X}}$ matrix X consisting of N vectors $x_n \in \mathcal{X}$ to a single condition $m(\psi; \cdot)$ to get an N-vector, i. e. $m(\psi; X) = (m(\psi; x_n))_{n=1}^{N} \in \{0, 1\}^N$.

(and thus metaheuristic operator-agnostic) edit distance; we count an edit only if it would change behaviour on the training data.

This can be defined formally using two measures at two levels: $\delta_X(\cdot,\cdot)$ measures the dissimilarity between *two particular rule conditions* whereas $d_X(\cdot,\cdot)$ combines these condition-wise dissimilarities into a dissimilarity measure over *sets* of conditions (i. e. over model structures). We start by explaining $\delta_X(\cdot,\cdot)$ which is parametrized with a given training set's input data points X. As was already explained above, we use interval-based conditions and a condition's $m(\psi;\cdot)$ parameter vector ψ therefore induces a ($\mathcal{D}_\mathcal{X}$-dimensional) interval $[l,u]$. In order to measure the dissimilarity between two such intervals, we *count the number of edits required to transform each of the bounds into the corresponding bound of the other interval*. At that, a single edit corresponds to moving one bound so that one more (or one less) training data point is included in the interval. We consider each dimension independently of the others (which is why this is an upper bound on the worst-case edit distance) because computing optimal sequences of edits is computationally infeasible. We can compute the edit count by considering each training data point independently and counting which interval bounds could *traverse* it. This *traversal count* can be formalized as (a visualization of this for two two-dimensional intervals is given in Fig. 1):

$$\delta_X(\psi_1,\psi_2) = \delta_X([l_1,u_1],[l_2,u_2]) \tag{7}$$

$$= \sum_{d=1}^{\mathcal{D}_\mathcal{X}} |\{x \in X \mid x \in H(\psi_1,\psi_2), L_{\min}(d) \le x_d \le L_{\max}(d)\}|^{\frac{1}{\mathcal{D}_\mathcal{X}}} \tag{8}$$

$$+ |\{x \in X \mid x \in H(\psi_1,\psi_2), U_{\min}(d) \le x_d \le U_{\max}(d)\}|^{\frac{1}{\mathcal{D}_\mathcal{X}}}$$

where $H(\psi_1,\psi_2) = \mathrm{Hull}([l_1,u_1],[l_2,u_2])$ is the convex hull (this arises naturally, see Fig. 1) of the two intervals and

$$L_{\min}(d) = \min(l_{1d}, l_{2d}), \qquad L_{\max}(d) = \max(l_{1d}, l_{2d}), \tag{9}$$

$$U_{\min}(d) = \min(u_{1d}, u_{2d}), \qquad U_{\max}(d) = \max(u_{1d}, u_{2d}). \tag{10}$$

Equation (8) is a sum that iterates over all input space dimensions and for each of them computes and sums the cardinality of two sets. The first set contains all data points $x \in X$ that lie in the interval's convex hull *and* between the lower of the two lower bounds of the two intervals in that dimension and the upper of the two lower bounds of the two intervals in that dimension. The second set analogously treats the intervals' upper bounds. The traversal count counts each $x \in X$ multiple times since it computes each dimension independently of the others which yields the aforementioned maximum number of traversals per data point. We further take the $\mathcal{D}_\mathcal{X}$th root of the set cardinalities in order to treat each dimension independently and compute a *per-dimension* instead of a per-volume value; without the $\mathcal{D}_\mathcal{X}$th root, conditions that match more training data points (i. e. larger intervals) would have a generally higher dissimilarity score just due to their higher volume.

This dissimilarity measure fulfills at the conditon-level (i. e. for model structures of size $K = 1$) the properties that we introduced above: Symmetry

(Property 1) is rather straightforward to show, Properties 2 and 3 follow directly from how the measure considers training data points being matched and we argue that Property 4 is fulfilled as well since the measure will continue to count training data points between conditions even if they do not overlap.

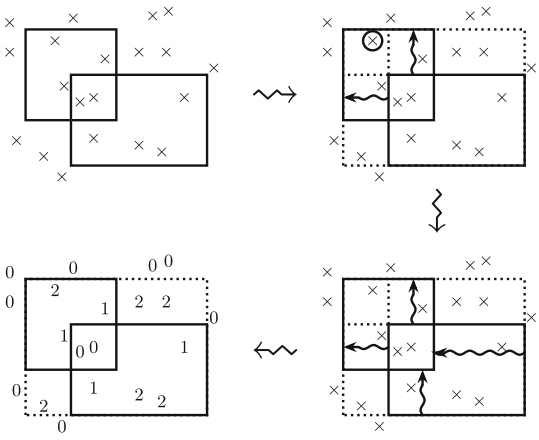

Fig. 1. Condition-wise dissimilarity measurement for two two-dimensional interval-based conditions. *Top left:* Data points as crosses, intervals as solid line rectangles (we call the square interval on the left ψ_1 and the non-square one on the right ψ_2). *Top right:* One data point x and the edit movements of the bounds that are relevant for possible traversals of x marked. If ψ_2's y-axis upper bound is transformed into ψ_1's y-axis upper bound first, then x is traversed when ψ_2's x-axis lower bound is transformed into ψ_1's x-axis lower bound. If ψ_2's x-axis lower bound is transformed into ψ_1's x-axis lower bound first, then x is traversed when ψ_2's y-axis upper bound is transformed into ψ_1's y-axis upper bound. This yields two possible traversals for x. *Bottom right:* All edit movements of this two-dimensional example. *Bottom left:* Data point crosses replaced by their respective number of traversals; as can be seen, only data points in the convex hull (dotted line) of the two intervals can ever be traversed and data points within the intersection of the intervals are never traversed.

In order to compute the dissimilarity between two model structures $\mathcal{M}_1$ and $\mathcal{M}_2$ (i. e. two *sets* of conditions), we compute the (mean) *sum of minimum distances*, which is a known set-wise dissimilarity measure (cf. e. g. [10]):

$$d_X(\mathcal{M}_1, \mathcal{M}_2) = \frac{1}{2}\left(\sum_{\psi_1 \in \mathcal{M}_1} \min_{\psi_2 \in \mathcal{M}_2} \delta_X(\psi_1, \psi_2) + \sum_{\psi_2 \in \mathcal{M}_2} \min_{\psi_1 \in \mathcal{M}_1} \delta_X(\psi_1, \psi_2)\right) \quad (11)$$

Note that only taking one of the sums (instead of their mean) would not yield a symmetric measure since minimizing dissimilarity may yield different results depending on which set of conditions is minimized over. Further, there are other options for combining the condition-wise dissimilarities into a set-wise dissimilarity; an overview of several is given by Eiter and Mannila [10].

The properties given in Sect. 4.1 are fulfilled for $d_X(\cdot,\cdot)$ as well, they carry over naturally from $\delta_X(\cdot,\cdot)$. As an aside, be aware that proving whether or not d is a *metric* (it is not yet clear whether it fulfills the triangle equality—we conjecture that it does not) is out of the scope of this paper.

Finally, it should be noted that computing $d_X(\mathcal{M}_1, \mathcal{M}_2)$ can be computationally expensive, especially if both $\mathcal{M}_1$ and $\mathcal{M}_2$ contain many rules. This is due to having to compute $\delta_X(\psi_1, \psi_2)$ for *all possible pairings* of $\psi_1 \in \mathcal{M}_1$ and $\psi_2 \in \mathcal{M}_2$ in order to compute the two summands in Eq. (11). At that, evaluating $\delta_X(\psi_1, \psi_2)$ can in itself be expensive for higher dimensions and more training data points.[3] In many cases, this is not that much of a problem, though, since the dissimilarity measure is meant mainly for post-hoc analysis and *not* for being evaluated during the investigated algorithm's runtimes.

5 Demonstration

To show the applicability of our dissimilarity measure, we apply it to the comparison of three different parametrizations each of DTs, RFs and XCSF.

5.1 Data-Generating Processes

We first generated a set of data-generating processes. Since the processes themselves are not relevant for the following demonstration of the dissimilarity measure, we will not go into detail here and refer the reader to our code [31]. We generated a range of learning tasks for dimensionalities $\mathcal{D}_\mathcal{X} \in \{3, 5, 8\}$, rule counts $K \in \{2, 4, 8, 10, 14, 18\}$ and two minimum coverage rates $\kappa_{\min} \in \{0.75, 0.9\}$. Sampling for these parameters produced a total of 132 learning tasks with an average of 3.77 tasks per configuration (and correspondingly 7.54 tasks if pooling over minimum coverage rates). From each learning task we generated training and test data sets by uniformly sampling inputs from the respective input space and based on those then sampling the random variable corresponding to outputs (see Eq. (3)). The number of training and test data points depended on the learning task's dimensionality: We generated $200 \cdot 10^{\mathcal{D}_\mathcal{X}/5}$ training data points and ten times as many test data points for each task, this yielded 796, 2000 and 7962 training data points for the three dimensionalities considered.

5.2 Evaluation of Repeated Runs

Next, we performed a set of experiments for which we used the DT and RF implementations provided by the *Scikit-learn* Python library [28] and the XCSF implementation provided by Preen's *XCSF* Python library [30]. We use three

[3] For $N = 768$ training data points, our own (not at all optimized) code took around 0.0005 s per computation of δ_X (mean over *all* computations of d_X with $N = 768$ performed for Fig. 2) and correspondingly around 0.2 s for computing d_X for two model structures of size 20. For $N = 2000$, we measured 0.002 s per δ_X computation (and correspondingly 0.8 s for size 20 model structures).

different parametrizations for each of the algorithm families by allowing a certain range for the maximum number of rules in the final model: up to 50, up to 100 and up to 200 rules (and 20 times as many for RFs). Further, all three algorithms are set to use constant models (for DTs and RFs this is already the default) and XCSF is set to use interval-based conditions.

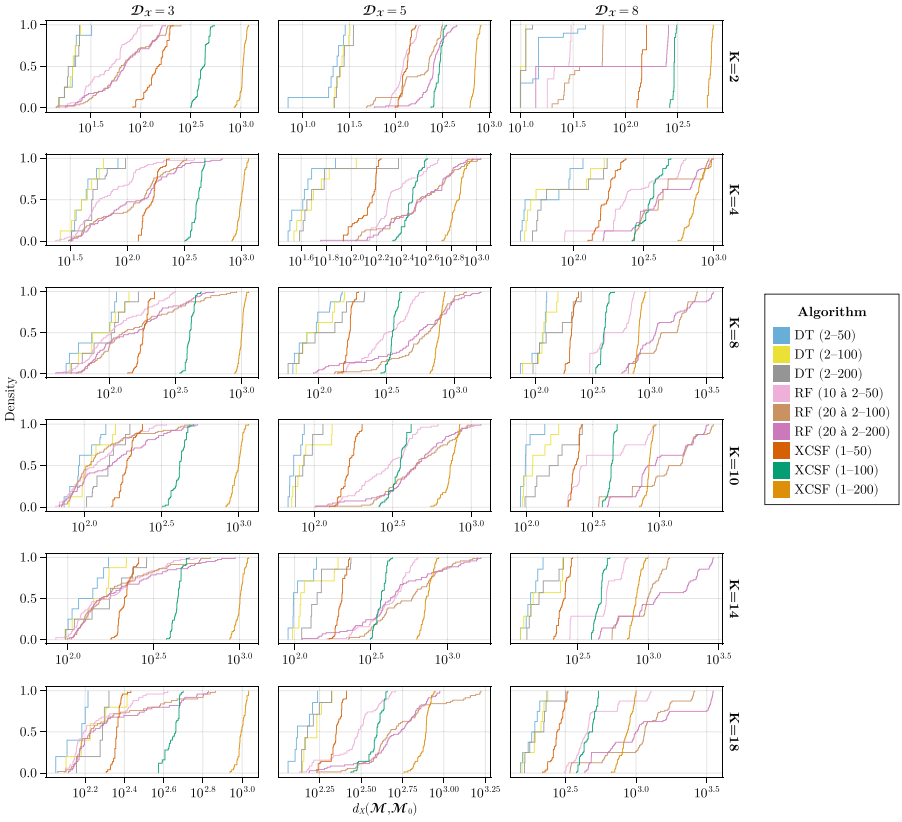

Fig. 2. Empirical cumulative distribution of $d(\cdot, \mathcal{M}_0)$ when pooling all the runs of each algorithm (i. e. pool over repetitions and minimum coverage rate values) for each combination of dimensionality $\mathcal{D}_\mathcal{X}$ (columns) and number of rules K (rows). Number ranges in the legend show the respective variant's range of allowed rule set sizes (in case of RF, a fixed number of used DTs and a range of allowed sizes for each).

We performed hyperparameter optimization on each of the 132 generated training data sets for each of the 9 algorithms used; we used the Optuna framework [1] to do so. Due to this paper's space constraints, we have to refer the reader to our code [32] for the parameter ranges and configurations used.

For each of the nine algorithm variants listed above, we then performed ten repetitions (ten consecutive random seeds) on each of the learning tasks

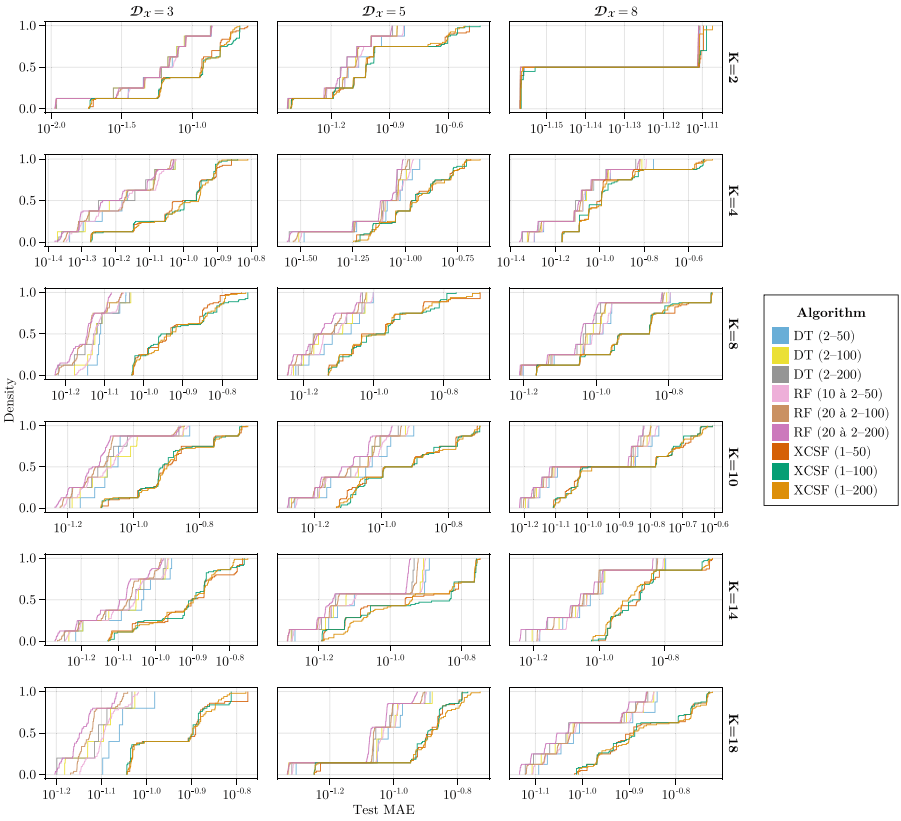

Fig. 3. Empirical cumulative distribution of test MAE. See Fig. 2 for details.

yielding an overall number $132 \times 9 \times 10 = 11880$ trials. Each final model is then evaluated for its predictive performance (MAE) using the holdout test set which was generated for the respective learning task. Further, we extract from each final model the set of rules $\mathcal{M}$ in interval representation and compute $d(\mathcal{M}, \mathcal{M}_0)$, its dissimilarity (see Eq. (11)) to the respective learning task's data-generating process's model structure $\mathcal{M}_0$. It should be noted at this point that a DT creates a set of non-overlapping rules that fully cover the input space, XCSF's rules are *allowed* to overlap, and an RF algorithm generates a fixed number d of DTs which means that there are at any one point in input space d rules.

Figure 2 shows the empirical cumulative distributions of $d(\cdot, \mathcal{M}_0)$ when pooling all the runs of each algorithm (i. e. pool over repetitions and minimum coverage rate values) for each combination of dimensionality $\mathcal{D}_\mathcal{X}$ (columns) and number of rules K (rows). In this figure, each graph in each diagram corresponds to one of the nine algorithm variants. This figure first of all shows that XCSF with a population size of 200 performs worse in terms of $d(\cdot, \mathcal{M}_0)$ than most of the other algorithms in most cases. In fact it only outperforms RFs containing

at least 20 DT for $\mathcal{D}_{\mathcal{X}} = 8$ and $K \geq 4$. Suprisingly, the three XCSF variants are spaced roughly proportional to the their maximum population sizes; upon closer investigation, this effect can be explained by the fact that the variants with maximum population sizes 50 and 100 exhaust that maximum fully in all cases and the variant with maximum population size 200 is only able to slightly reduce the number of rules in some cases using the subsumption mechanism. The DTs better performance is mainly due to creating models with an overall much smaller number of rules (mean±std, rounded to one decimal, 10.3±4.9, 13.4±7.4 and 16.3±9.8 rules for the three DT variants) which means that there are much fewer summands in Eq. (11). The RFs, on the other hand, perform worse than the XCSF variants on a subset of the learning tasks because they have a larger number of rules than XCSF (yielding more summands in Eq. (11)).

When comparing Fig. 2 which shows $d(\cdot, \mathcal{M}_0)$ with Fig. 3 which shows MAE we can see that evaluating for MAE only separates the tree-based algorithms from the XCSF variants while $d(\cdot, \mathcal{M}_0)$ provides more nuance. It is interesting to note here that the RFs seem to perform better relative to the other algorithms with respect to test MAE but not with respect to $d(\cdot, \mathcal{M}_0)$. One possible explanation is the performance-interpretability tradeoff: RFs use more rules than DTs and XCSF (worse interpretability) but at the same time the increase in parameters results in them being able to model the data better.

6 Related Work

This section discusses some more related work that has not yet been mentioned in the preceding text.

In some sense the present work can be seen as being in the same spirit as the work that Kovacs and Kerber did in the early 2000 s where they questioned the common practice that performance of Learning Classifier Systems (LCSs, which are a prominent member of the MRSL family), most notably XCS, was measured using accuracy (or, more generally, error) alone [20,21]. They argued that this does not fully capture the actual target one has in mind when thinking about the performance of these systems.

To circumvent the weaknesses of only measuring predictive performance and more directly assess progress of the metaheuristic, Kovacs defined several alternative measures [19,20] based on comparing the sets of rules to some form of known optimal solution. However, Kovacs's work was restricted to binary classification tasks with binary vector inputs and transfer to regression tasks does not seem possible. That aside, the used notion of optimal rule sets was informed by already observed behaviour of the systems to be investigated [20] whereas we try hard to stay algorithm-agnostic with our approach. Another weakness of Kovacs's measures of closeness to the data-generating process is that they only check *how many* of the rules in the optimal rule set the final rule set contains. However, *extracting* optimal rules from a rule set that contains them is not solved yet in general (there are merely some heuristics available, each with strengths and weaknesses, that try to do that [25]). Other authors agree with us on that,

e. g. Drugowitsch [9] writes in 2008 "[studies by Kovacs et al.] aimed at defining the optimal set for limited classifier representations [but there] was still no general definition available" and then continues to derive a probabilistic framework for defining rule set fitness from first principles. However, that framework is, as of now, computationally infeasible even for low input space dimensionalities.

Tan et al. look at different rule compaction approaches and in order to understand their difference, they compute similarity scores of the rule sets that these different approaches generate [39]. Their score is computed as the *ratio of rules preserved* and is thus not symmetric (Property 1).

Heider et al. [16] investigate novelty search for discovering rules in an MRSL algorithm called SupRB. In order to measure a candidate rule's novelty (i. e. a form of dissimilarity), they use the Hamming distance to compare its match set to the matching vectors of its nearest neighbours. However, this measure assigns to two rules that do not overlap a dissimilarity value that does not change with spatial distance but only with the rules' volumes (Property 4).

Setnes et al. [36] consider compacting models generated by fuzzy rule-based systems. They define similarity between two rules using the fuzzy set-equivalent to the rules' match sets and therefore there is, again, no differentiation between rule positions as soon as rules do not overlap anymore (Property 4).

Kharbat et al. [18] perform for binary classification with binary vector inputs a form of rule set compaction based on clustering rules. They define rule similarity using the Jaccard binary similarity coefficient which also shares the problem of not differentiating between non-overlappying rules (Property 4).

[11] propose a dissimilarity measure between *data sets* which is based on training DTs on the data sets and then computing a dissimilarity between the resulting models. In order to compare two DTs, they compute a third DT (they call this the *greatest common refinement*) which contains all the feature splits from the two base DTs and allows to compare the DTs' predictions region-by-region. However, comparing predictions is not the target of our work.

Serpen and Sabhnani [35] compare two rule sets like the ones we consider by computing their respective volumes as well as the volume of their overlap. However, volume alone is too weak a signal to investigate MRSL metaheuristics; in particular, this does not fulfill Property 4.

Earlier RSL literature explored heuristical detection of changes in data distributions. Some of these approaches (e. g. [22–24,29,40]) fit models to the data repeatedly and then compare these models in order to detect changes in the data distributions. However, since these approaches naturally assume the rate of change is small, their dissimilarity measures are not suitable for models that may be fundamentally different (e. g. Property 4). Aside from that, they handle only classification tasks most of the time.

Edit distance (sometimes also called *mutation distance*) has been studied for improving Genetic Programming algorithms; for example, Gustafson and Vanneschi [13] give an overview of approaches and propose a crossover-based edit distance measure. However, distances between trees are fundamentally different from distances between sets of intervals that are allowed to overlap.

Finally, while there have been proposed for metaheuristics many similarity measures (cf. [4,6]) as well as diversity measures (e. g. [8]) that often also measure similarity, to our knowledge, none of the currently available ones consider phenotypes that correspond to sets of intervals and even less so intervals that correspond to the conditions of rules.

7 Future Work

The main segment of future work will be to *use* our proposed dissimilarity metric to benchmark different variants of existing MRSL algorithms, analyse their models and based on the findings develop for these systems new and better-performing metaheuristic operators (or combinations thereof). Aside from that, we conjecture that the measure is suited for a hyperparameter tuning regime: Given an unknown learning task, one could capture one's knowledge of and beliefs about it (e. g. input space dimensionality, suspected number of rules that can approximate the task well, distribution over condition volumes etc.) as a set of *helper tasks*—a set of rule-set-based data-generating processes [27] with matching properties. One could then minimize on the helper tasks an MRSL algorithm's hyperparameters with respect to the dissimilarity measure in order to obtain a hyperparametrization for the unknown learning task.

Apart from measuring dissimilarity to a known data-generating process, the measure may prove useful to be included *within* population-based MRSLs as a measure of similarity between solutions (e. g. to build mechanisms that encourage population diversity, increase exploration and reduce preliminary convergence).

We chose the sum of minimum distances in Eq. (11) for now for its simplicity and the ease with which one can reason about it but intend to implement and test Eiter and Mannila's alternative, the link distance [10], as well.

As noted above, computing the dissimilarity measure can be expensive; however, large parts of that computation can be parallelized which should yield significant speedups on now-typical hardware.

8 Conclusion

We proposed a way to measure dissimilarity between rule sets created by regression *Metaheuristic Rule Set Learners* (MRSLs). Dissimilarity between two rules is defined as a lower bound on the maximum number of prediction-changing edits required to transform one rule into the other; this is extended to sets of rules using a well-known set distance measure. The measure tries to capture better the preferences of a user using MRSL algorithms in scenarios where interpretability is aimed at (i. e. a few well-placed rules are preferred over the highest possible performance). Our dissimilarity measure provides an additional target for MRSL development that allows to measure metaheuristic progress more directly than would be possible using predictive error-based measures such as mean absolute error. We expect that the presented tooling helps in overcoming the percieved decline in research activity on MRLSs by allowing to better analyse the metaheuristic's effects on model structure.

References

1. Akiba, T., Sano, S., Yanase, T., Ohta, T., Koyama, M.: Optuna: a next-generation hyperparameter optimization framework. In: Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (2019)
2. Alcala, R., Gacto, M.J., Herrera, F.: A fast and scalable multiobjective genetic fuzzy system for linguistic fuzzy modeling in high-dimensional regression problems. IEEE Trans. Fuzzy Syst. **19**(4), 666–681 (2011). https://doi.org/10.1109/TFUZZ.2011.2131657
3. Bernadó-Mansilla, E., Garrell-Guiu, J.M.: Accuracy-based learning classifier systems: models, analysis and applications to classification tasks. Evolut. Comput. **11**(3), 209–238 (2003). https://doi.org/10.1162/106365603322365289
4. Brusco, M., Cradit, J.D., Steinley, D.: A comparison of 71 binary similarity coefficients: the effect of base rates. Plos One **16**(4) (2021)
5. Butz, M.V., Stolzmann, W.: An algorithmic description of ACS2. In: Lanzi, P.L., Stolzmann, W., Wilson, S.W. (eds.) IWLCS 2001. LNCS (LNAI), vol. 2321, pp. 211–229. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-48104-4_13
6. Choi, S.S., Cha, S.H., Tappert, C.C.: A survey of binary similarity and distance measures. J. Syst. Cybernet. Inform. **8**(1), 43–48 (2010)
7. Cordón, O.: A historical review of evolutionary learning methods for mamdani-type fuzzy rule-based systems: designing interpretable genetic fuzzy systems. Int. J. Approximate Reasoning **52**(6), 894–913 (2011). https://doi.org/10.1016/j.ijar.2011.03.004
8. Corriveau, G., Guilbault, R., Tahan, A., Sabourin, R.: Review and study of genotypic diversity measures for real-coded representations. IEEE Trans. Evol. Comput. **16**(5), 695–710 (2012). https://doi.org/10.1109/TEVC.2011.2170075
9. Drugowitsch, J.: Design and Analysis of Learning Classifier Systems - A Probabilistic Approach. SCI, vol. 139. Springer, Berlin (2008). https://doi.org/10.1007/978-3-540-79866-8
10. Eiter, T., Mannila, H.: Distance measures for point sets and their computation. Acta Informatica **34**(2), 109–133 (1997). https://doi.org/10.1007/S002360050075
11. Ganti, V., Gehrke, J., Ramakrishnan, R.: A framework for measuring changes in data characteristics. In: Proceedings of the Eighteenth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 1999 pp. 126–137. Association for Computing Machinery, New York (1999). https://doi.org/10.1145/303976.303989
12. Gao, X., Xiao, B., Tao, D., Li, X.: A survey of graph edit distance. Pattern Anal. Appl. **13**(1), 113–129 (2010). https://doi.org/10.1007/S10044-008-0141-Y
13. Gustafson, S., Vanneschi, L.: Crossover-based tree distance in genetic programming. IEEE Trans. Evol. Comput. **12**(4), 506–524 (2008). https://doi.org/10.1109/TEVC.2008.915993
14. Heider, M., Pätzel, D., Stegherr, H., Hähner, J.: A Metaheuristic Perspective on Learning Classifier Systems, pp. 73–98. Springer Nature Singapore, Singapore (2023). https://doi.org/10.1007/978-981-19-3888-7_3
15. Heider, M., Stegherr, H., Nordsieck, R., Hähner, J.: Learning classifier systems for self-explaining socio-technical-systems (2022)
16. Heider, M., et al.: Discovering rules for rule-based machine learning with the help of novelty search. SN Comput. Sci. **4**(6), 778 (2023). https://doi.org/10.1007/s42979-023-02198-x

17. Heider, M., Stegherr, H., Wurth, J., Sraj, R., Hähner, J.: Separating rule discovery and global solution composition in a learning classifier system. In: Proceedings of the Genetic and Evolutionary Computation Conference Companion, GECCO 2022, pp. 248–251. Association for Computing Machinery, New York(2022). https://doi.org/10.1145/3520304.3529014
18. Kharbat, F., Odeh, M., Bull, L.: New approach for extracting knowledge from the XCS learning classifier system. Inter. J. Hybrid Intell. Syst. **4**, 49–62 (2007). https://doi.org/10.3233/HIS-2007-4201
19. Kovacs, T.: Deletion schemes for classifier systems. In: Proceedings of the 1st Annual Conference on Genetic and Evolutionary Computation, pp. 329–336 (1999)
20. Kovacs, T.: What should a classifier system learn and how should we measure it? Soft. Comput. **6**(3), 171–182 (2002)
21. Kovacs, T., Kerber, M.: High classification accuracy does not imply effective genetic search. In: Deb, K. (ed.) GECCO 2004. LNCS, vol. 3103, pp. 785–796. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-24855-2_93
22. Liu, B., Hsu, W., Han, H.-S., Xia, Y.: Mining changes for real-life applications. In: Kambayashi, Y., Mohania, M., Tjoa, A.M. (eds.) DaWaK 2000. LNCS, vol. 1874, pp. 337–346. Springer, Heidelberg (2000). https://doi.org/10.1007/3-540-44466-1_34
23. Liu, B., Hsu, W., Ma, Y.: Discovering the set of fundamental rule changes. In: Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD 2001, pp. 335–340. Association for Computing Machinery, New York (2001). https://doi.org/10.1145/502512.502561
24. Liu, B., Ma, Y., Lee, R.: Analyzing the interestingness of association rules from the temporal dimension. In: Proceedings 2001 IEEE International Conference on Data Mining, pp. 377–384 (2001). https://doi.org/10.1109/ICDM.2001.989542
25. Liu, Y., Browne, W.N., Xue, B.: A comparison of learning classifier systems' rule compaction algorithms for knowledge visualization. ACM Trans. Evol. Learn. Optim. **1**(3) (2021). https://doi.org/10.1145/3468166
26. Parpinelli, R.S., Lopes, H.S., Freitas, A.A.: An ant colony algorithm for classification rule discovery. In: Data Mining, pp. 191–208. IGI Global (2002). https://doi.org/10.4018/978-1-930708-25-9.ch010
27. Pätzel, D., Heider, M., Hähner, J.: Towards principled synthetic benchmarks for explainable rule set learning algorithms. In: Proceedings of the Companion Conference on Genetic and Evolutionary Computation, GECCO 2023 Companion, pp. 1657–1662. Association for Computing Machinery, New York (2023). https://doi.org/10.1145/3583133.3596416
28. Pedregosa, F., et al.: Scikit-learn: machine learning in Python. J. Mach. Learn. Res. **12**, 2825–2830 (2011)
29. Pekerskaya, I., Pei, J., Wang, K.: Mining changing regions from access-constrained snapshots: a cluster-embedded decision tree approach. J. Intell. Inf. Syst. **27**(3), 215–242 (2006). https://doi.org/10.1007/S10844-006-9951-9
30. Preen, R.J., Pätzel, D.: Xcsf (2023). https://doi.org/10.5281/zenodo.8193688
31. Pätzel, D.: dpaetzel/rslmodels.jl: v0.1.1. https://doi.org/10.5281/zenodo.10557400
32. Pätzel, D.: dpaetzel/run-rsl-bench: v1.1.0. https://doi.org/10.5281/zenodo.10550923
33. Pätzel, D.: dpaetzel/syn-rsl-benchs: v1.0.0 (May 2023). https://doi.org/10.5281/zenodo.7919420
34. Quinlan, J.R.: C4.5: Programs for Machine Learning. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (1993)

35. Serpen, G., Sabhnani, M.: Measuring similarity in feature space of knowledge entailed by two separate rule sets. Knowl.-Based Syst. **19**(1), 67–76 (2006). https://doi.org/10.1016/j.knosys.2003.11.001
36. Setnes, M., Babuska, R., Kaymak, U., van Nauta Lemke, H.: Similarity measures in fuzzy rule base simplification. IEEE Trans. Syst. Man Cybernet. Part B (Cybernetics) **28**(3), 376–386 (1998). https://doi.org/10.1109/3477.678632
37. Stalph, P.O., Butz, M.V.: Guided evolution in XCSF. In: Proceedings of the 14th Annual Conference on Genetic and Evolutionary Computation, GECCO 2012, pp. 911–918. Association for Computing Machinery, New York (2012). https://doi.org/10.1145/2330163.2330289
38. Tamee, K., Bull, L., Pinngern, O.: Towards clustering with XCS. In: Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation, GECCO 2007, pp. 1854–1860. Association for Computing Machinery, New York (2007). https://doi.org/10.1145/1276958.1277326
39. Tan, J., Moore, J.H., Urbanowicz, R.J.: Rapid rule compaction strategies for global knowledge discovery in a supervised learning classifier system. In: Liò, P., Miglino, O., Nicosia, G., Nolfi, S., Pavone, M. (eds.) Proceedings of the Twelfth European Conference on the Synthesis and Simulation of Living Systems: Advances in Artificial Life, ECAL 2013, Sicily, Italy, 2–6 September 2013, pp. 110–117. MIT Press (2013). https://doi.org/10.7551/978-0-262-31709-2-CH017
40. Wang, K., Zhou, S., Fu, C.A., Yu, J.X.: Mining Changes of Classification by Correspondence Tracing, pp. 95–106. https://doi.org/10.1137/1.9781611972733.9
41. Wilson, S.W.: Classifier fitness based on accuracy. Evol. Comput. **3**(2), 149–175 (1995)
42. Wilson, S.W.: Classifiers that approximate functions. Nat. Comput. **1**(2), 211–234 (2002). https://doi.org/10.1023/A:1016535925043