Instance Space Analysis for Algorithm Testing: Methodology and Software Tools

KATE SMITH-MILES, The University of Melbourne, Australia
MARIO ANDRÉS MUÑOZ, The University of Melbourne, Australia

Instance Space Analysis (ISA) is a recently developed methodology to (i) support objective testing of algorithms, and (ii) assess the diversity of test instances. Representing test instances as feature vectors, the ISA methodology extends Rice's 1976 Algorithm Selection Problem framework to enable visualization of the entire space of possible test instances, and gain insights into how algorithm performance is affected by instance properties. Rather than reporting algorithm performance on average across a chosen set of test problems, as is standard practice, the ISA methodology offers a more nuanced understanding of the unique strengths and weaknesses of algorithms across different regions of the instance space that may otherwise be hidden on average. It also facilitates objective assessment of any bias in the chosen test instances, and provides guidance about the adequacy of benchmark test suites. This paper is a comprehensive tutorial on the ISA methodology that has been evolving over several years, and includes details of all algorithms and software tools that are enabling its worldwide adoption in many disciplines. A case study comparing algorithms for university timetabling is presented to illustrate the methodology and tools.

CCS Concepts: • **Theory of computation** → **Design and analysis of algorithms**; • **Information systems** → **Decision support systems**; • **Mathematics of computing** → *Statistical software*; • **General and reference** → *Empirical studies*; *Experimentation*; *Performance*.

Additional Key Words and Phrases: Algorithm footprints, Algorithm selection, Benchmarking, MATLAB, Meta-learning, Meta-heuristics, Software as a service, Test instance diversity, Timetabling.

1 INTRODUCTION

A significant pitfall affecting the validity and reliability of algorithm testing is the adoption of benchmarking test instances which fail to meet necessary diversity criteria. Ideally, any suite of test instances used to establish trust in an algorithm should be unbiased, challenging, and contain a mix of synthetically generated and real-world-like instances with sufficiently diverse structural properties to enable the strengths and weaknesses of algorithms to be exposed. Given that the conclusions drawn during benchmarking critically depend on the choice of test instances [17], without such diversity the trustworthiness of an algorithm for future untested instances is necessarily limited.

In many fields, the standard benchmarking practice involves testing algorithms on well-studied test suites, inherited from the literature, without necessarily scrutinizing their diversity or suitability [13, 16, 24, 26]. Moreover, standard experimental benchmarking practice focuses on reporting on-average performance across these collections of well-studied test instances [25], defining algorithm success if it has a superior performance on average compared to other algorithms [16, 17]. The weaknesses of an algorithm are rarely reported, and how

Authors' addresses: Kate Smith-Miles, smith-miles@unimelb.edu.au, ARC Training Centre in Optimisation Technologies, Integrated Methodologies and Applications (OPTIMA), School of Mathematics and Statistics, The University of Melbourne, Parkville, Victoria, 3010, Australia; Mario Andrés Muñoz, munoz@unimelb.edu.au, ARC Training Centre in Optimisation Technologies, Integrated Methodologies and Applications (OPTIMA), School of Computer and Information Systems, The University of Melbourne, Parkville, Victoria, 3010, Australia.

0360-0300/2022/11-ART $15.00
https://doi.org/10.1145/3572895

the performance depends on the properties of the test instances is obscured by such on-average reporting. As such, the standard benchmarking practice has two limitations that directly impact the establishment of trust in algorithms: (a) there is no mechanism to assess whether the selected test instances are unbiased and diverse enough to support the broadest possible conclusions; and (b) there is little opportunity to gain insights into the strengths and weaknesses of algorithms for different types of instances, when hidden by on-average performance metrics.

Instance Space Analysis (ISA) is a recent methodology that offers a paradigm shift in the way algorithms are evaluated, revealing insightful relationships between the structural properties of test instances and their impact on the performance of algorithms. By constructing an *instance space* containing all possible test instances in a $2D$ plane, ISA essentially “cracks open" the suite of test instances, creating opportunities to gain more nuanced insights into algorithm strengths and weaknesses for various types of test instances. The outcome is a more objective assessment of the relative power of algorithms, when supported by a demonstrably unbiased and diverse choice of test instances.

First proposed by Smith-Miles and co-workers [38], the ISA methodology has evolved through a series of advances [18, 19, 27–30, 39] into its current form and associated software tools described in this paper. The theoretical foundations of ISA can be traced to the Algorithm Selection Framework developed by J. Rice in 1976 [34], which proposed using features of test instances to predict algorithm performance. We refer the interested reader to the survey paper by Smith-Miles [37] which reviewed progress on the Algorithm Selection Problem and pointed to opportunities for other fields to benefit from the framework. The No-Free Lunch theorems of Wolpert and Macready [47] in 1997 provided further theoretical foundations and support for the idea to build upon such concepts to improve insights into algorithm strengths and weaknesses. ISA extends the framework provided by Rice by constructing a $2D$ instance space to expose the similarity and differences between chosen test instances based on a set of features that characterize their difficulty for algorithms. The chosen test instances are realizations of the broader set of all possible test instances, and the mathematical boundary defining this theoretical instance space can be derived and visualized within the $2D$ plane. Scrutinizing the performance of algorithms on the chosen test instances enables machine learning algorithms to predict regions where good performance can be expected, based on empirical evidence provided by the chosen test instances as training data. The area of the instance space where an algorithm is predicted to demonstrate good performance is defined as the *algorithm footprint*, and provides an objective measure of the relative power of algorithms compared to each other, as well as their applicability and robustness. From this view of a broad instance space, we can also assess the diversity and potential bias of a chosen set of test instances, and gain much needed insights into how structural properties of various instances influence the footprints describing the strengths and weaknesses of algorithms. Moreover, through ISA we can identify where additional test instances would be valuable to support greater insights. By setting target points in the instance space, new test instances with controllable properties can be generated to fill the instance space, enabling algorithms to be comprehensively “stress-tested” under all theoretically possible conditions [27, 29, 39].

This paper is a tutorial that uses a case study from university course timetabling to illustrate and summarize:

(1) the ISA methodology – supporting objective algorithm testing and scrutiny of test instance diversity;
(2) the ISA Toolkit – a MATLAB-based set of tools that performs ISA automatically [32];
(3) MATILDA – the Melbourne Algorithm Test Instance Library with Data Analytics (MATILDA) cloud tools for online analysis [43].

Both MATILDA and the ISA Toolkit allow researchers to apply the ISA methodology for new problems, analyze new algorithms, assess the adequacy of existing benchmark test suites, and carry out automated algorithm selection, without engaging with the source code. Moreover, MATILDA also provides a collection of library problems with successful ISA results, and open-source meta-data, for several well-studied optimization, learning and model fitting problems, as case studies from previously published research. The intention is for this library of case studies to expand over time as more researchers adopt ISA to benchmark algorithms, and share their test instances, algorithms and instance feature code.

The remainder of this paper continues as follows: After presenting the mathematical formalism of the ISA framework in Section 2, we present details of the methodology and the core components of the analysis pipeline in Section 3. A case study is offered in Section 4 —from the university course curriculum timetabling problem— to demonstrate the kinds of insights that can be offered through ISA, and how the conclusions drawn about algorithm performance are more nuanced and insightful than on-average statistical analysis of suites of benchmark test instances. Section 5 concludes the paper by discussing the opportunities offered by the ISA methodology and freely available tools, as well as avenues for further research to expand its capabilities.

2 INSTANCE SPACE ANALYSIS: CONCEPTUAL FRAMEWORK

The conceptual framework underpinning ISA is illustrated in Figure 1, showing how an extension to Rice's Algorithm Selection Problem [34], can support greater insights beyond predicting a winning algorithm [37]. At its core, there are six component *spaces* or sets. The first is the ill-defined *problem space*, $\mathcal{P}$, containing all the relevant instances of a problem in an application domain. For example, in the university course curriculum timetabling problem considered as a case study in Section 4, the problem space contains all possible test instances from any university in any year. From $\mathcal{P}$, we have a *subset of instances*, $\mathbf{I}$, for which we have data from computational experiments. This experimental data is used to generate *meta-data* about the available instances, their difficulty and their characteristics. The first component of the meta-data are some chosen *features*, or hardness metrics, used to characterize the mathematical and statistical properties of the instances that: (a) describe the similarities and differences between instances in $\mathbf{I}$; and (b) have correlation with the performance of one or more algorithms. The features are calculated for a given instance x, and comprise the elements of the feature vector, $\mathbf{f}_x$, which represents an instance in the *feature space*, $\mathcal{F}$.

The second component of the meta-data is a *performance measure* for each algorithm α when solving an instance $x \in \mathbf{I}$, where α is an element of the *algorithm space*, $\mathcal{A}$, representing the set of algorithms available to solve all instances in $\mathbf{I}$. This performance measure, $y_{\alpha,x}$, is an element of a vector that describes the *performance space*, $\mathcal{Y}$, and requires a user-defined measure of "goodness", such as the computational effort to obtain a satisfactory solution, or the solution quality obtained for a fixed computational budget. Both the features and performance measures for all the instances in $\mathbf{I}$, and all algorithms in $\mathcal{A}$ constitute the meta-data, which are represented as two matrices $\mathbf{F} = [\mathbf{f}_1 \ \ldots \ \mathbf{f}_n] \in \mathbb{R}^{m\times n}$ and $\mathbf{Y} = [\mathbf{y}_1 \ \ldots \ \mathbf{y}_n] \in \mathbb{R}^{a\times n}$, where m is the number of features, n is the number of problem instances, and a is the number of algorithms.

Within the original Algorithm Selection Problem framework of Rice [34], shown in the lower half of Figure 1, the meta-data is used to learn a selection mapping, $S(\cdot)$, typically using regression as a prediction model of algorithm performance based on instance features, which is used to identify the algorithm α^* most likely to be best for a given instance x: $\alpha^* = \arg\max \ S\left(\mathbf{f}_x, y_{\alpha,x}\right)$. ISA extends this framework into the upper half of Figure 1, by using the same meta-data for an additional goal: to map each instance from its m-dimensional feature vector to a point $\mathbf{z}$ in a $2D$ *instance space*. A customized dimension reduction method projects the instances within a new $2D$ coordinate system, based on an automated feature selection process, designed to emphasize the observable linear trends across the instance space when the distribution of algorithm performance measures and features are inspected. In this way, regions of the instance space can be associated as easy or hard for different algorithms, and the features of the instances lying to those regions can be identified, thus supporting insights into how instance characteristics affect algorithm performance. Moreover, machine learning methods can also learn the relationship between the coordinates of an instance in the $2D$ instance space and algorithm performance, thus providing automated algorithm selection, like Rice's approach, but with the additional advantage of visualization.

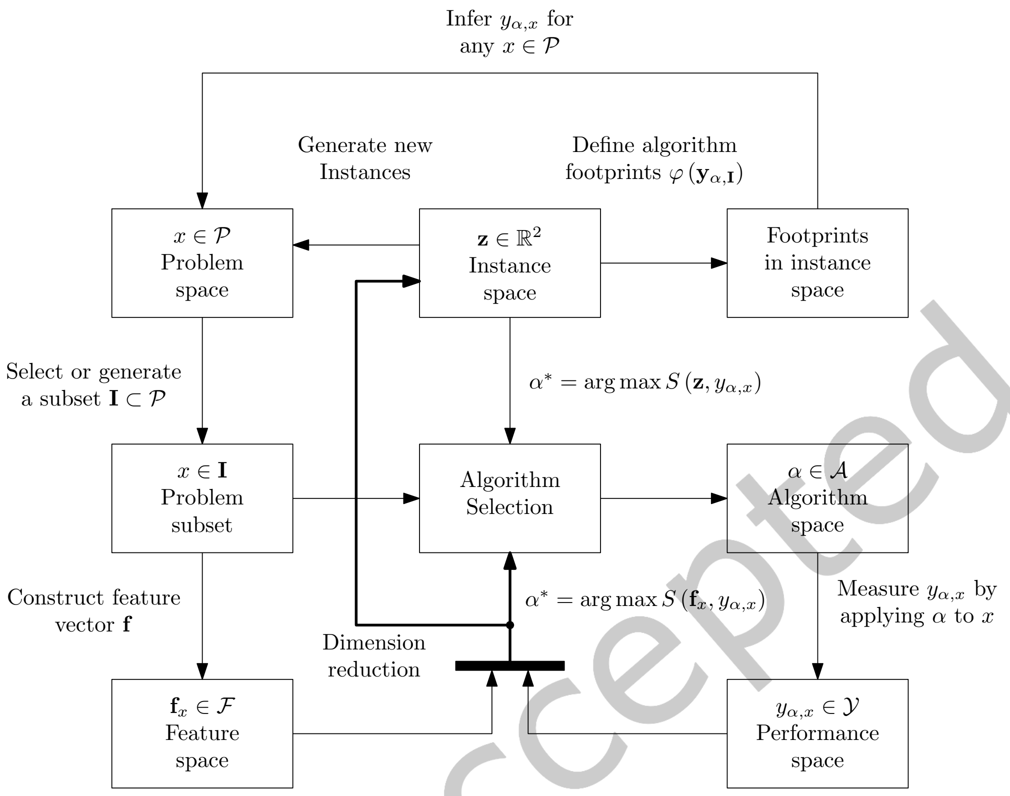

Infer $y_{\alpha,x}$ for any $x \in \mathcal{P}$
Generate new Instances
Define algorithm footprints $\varphi(\mathbf{y}_{\alpha,\mathbf{I}})$
$x \in \mathcal{P}$ Problem space
$\mathbf{z} \in \mathbb{R}^2$ Instance space
Footprints in instance space
Select or generate a subset $\mathbf{I} \subset \mathcal{P}$
$\alpha^* = \arg\max S(\mathbf{z}, y_{\alpha,x})$
$x \in \mathbf{I}$ Problem subset
Algorithm Selection
$\alpha \in \mathcal{A}$ Algorithm space
Construct feature vector $\mathbf{f}$
$\alpha^* = \arg\max S(\mathbf{f}_x, y_{\alpha,x})$
Measure $y_{\alpha,x}$ by applying α to x
Dimension reduction
$\mathbf{f}_x \in \mathcal{F}$ Feature space
$y_{\alpha,x} \in \mathcal{Y}$ Performance space

Fig. 1. Summary of the ISA framework

An instance space perspective of the meta-data enables additional insights into the adequacy of test instances. In addition to projecting the instances to the $2D$ plane, the upper and lower bounds of the features enable a bounding region to also be projected from the higher dimensional feature space to the instance space, thereby creating a mathematically defined theoretical boundary, beyond which no instance can theoretically exist. This boundary is critical to assess the diversity of the test instances in the meta-data.

The instance space also offers a more nuanced assessment of the unique strengths and weaknesses of algorithms, compared to standard on-average statistical reporting. Comparisons between algorithms are performed by estimating their *footprints*, $\varphi(\mathbf{y}_{\alpha,\mathbf{I}})$, which are the regions in the instance space where we statistically infer good performance of the algorithm, using machine learning methods, for a user-defined criteria of goodness. The properties of a footprint can be assessed to draw conclusions about algorithm power in absolute terms, as well as relative to other algorithms. These footprint properties are its *location*, which determines the type of instances (e.g. real-world), where good performance can be expected; *area*, which provides a measure of algorithm robustness; *density*, which indicates the statistical strength of evidence; and *purity*, which indicates whether there is conflicting evidence or not. In other words, an algorithm is deemed *powerful* if there is substantial evidence that it performs well in a large and dense region of the instance space, with little or no conflicting evidence. If an algorithm's footprint is unique compared to other algorithms, or if the footprint encompasses interesting classes of instances, such as real-world instances, or challenging instances, we can articulate the strengths of the algorithm in a more meaningful manner than merely reporting that it is best on average. The distribution of features across the instance space and within footprints can be used to gain insights into the kinds of instances that are well suited to each algorithm's strengths, and those that create challenges and expose weaknesses. Moreover, by partitioning the instance space into unique algorithm footprints, ISA also supports automated algorithm selection, enabling the recommendation of the most-likely best performing algorithm to be selected for a given instance based on its location in the space.

The ISA methodology is an iterative process, whereby an initial instance space is created and explored based on currently available meta-data $\{\mathbf{F}, \mathbf{Y}\}$. The outcome of the first iteration may be identification of gaps in the instance space where new test instances need to be generated at target locations [39], or where the current set of features needs to be augmented to find features that are more discriminating of variations in algorithm performance. Once new instances are added to $\mathbf{I}$, different features may be selected to best describe relationships to algorithm performance, and the $2D$ axes defining the projected instance space are likely to change as well. The entire process can be repeated until convergence, when the generated instances fully occupy the interior of the instance space boundary, and the features explain variations in algorithm performance is an insightful manner.

3 INSTANCE SPACE ANALYSIS: METHODOLOGY

Within the conceptual framework outlined in the previous section, there are six core steps to the ISA methodology:

(1) Collect experimental meta-data for a set of instances ($\mathbf{I}$) run on a portfolio of algorithms ($\mathcal{A}$), comprising feature values ($\mathbf{F}$) for all instances, and a performance metric ($\mathbf{Y}$) for all algorithms on all instances;
(2) Construct an instance space using a feature selection process on the meta-data $\{\mathbf{F}, \mathbf{Y}\}$, for a user-defined measure of good performance, including its theoretical boundary;
(3) Generate machine learning predictions for automated algorithm selection;
(4) Generate algorithm footprints and metrics;
(5) Analyze the instance space to (i) gain insights based on current meta-data; (ii) review the sufficiency of the meta-data;
(6) Generate additional meta-data if required, and repeat from Step 2; else stop.

The initial meta-data may be insufficient to gain the desired insights into how instance characteristics explain algorithm performance. Specifically, analysis of the instance space may reveal that the instances are not diverse enough, or are biased towards a limited region of the instance space; the algorithms' footprints may be too similar, or dominated by one algorithm due to the noncompetitive nature of the portfolio, or fail to perform well in some regions of the instance space where other algorithms may be more suitable; and the performance of algorithms may be too hard to predict in some regions where too many contradictions exists, due to the chosen features failing to capture the intrinsic difficulty of some instances. In all of these cases, ISA reveals the opportunity to augment the meta-data to improve its potential for greater insights. In Step 6, we enrich the meta-data by identifying where new test instances are required to fill gaps[1], whether additional algorithms are required to cover the instance space, and if new features are required to better explain algorithm performance. Convergence of the methodology is achieved when: (i) there are no remaining gaps or holes between instances, which are sufficiently diverse and dense enough to fill the interior of the instance space boundary; and (ii) the features explain well the performance of algorithms, and additional proposed features are not selected as components of the projection mapping.

In the following sections, we present implementation details of each of these core steps in the ISA methodology, including key methods and algorithms within the ISA toolkit.

[1]While we have proposed methods for evolving new test instances to lie at target locations in the instance space using genetic algorithms [29, 39, 40], suitable encoding of an instance within a genetic algorithm is quite problem specific, and is therefore not included within the automated ISA toolkit discussed in this paper.

```
Instances, Source, feature_clusteringIndexDistTeacherMean, ... , algo_SACP, algo_TSCS
395Events90PctOccupancy-8.cttLopesSmithMiles, real-world like, 0.744186047, ..., 18, 1
323Events90PctOccupancy-16.cttLopesSmithMiles, real-world like, 0.75,1052, ..., 5, 0
347Events90PctOccupancy-3.cttLopesSmithMiles, real-world like, 0.736842105, ..., 5, 0
293Events90PctOccupancy-35.cttLopesSmithMiles, real-world like, 0.734042553, ..., 4, 0
...
comp21.ctt, competition, 0.117021277, ..., 124, 111
```

Fig. 2. Header for the meta-data file for the Timetabling problem available at MATILDA's website [43]. On the first line are the column identifiers, with subsequent rows each representing an instance.

3.1 Collecting meta-data

The collection of meta-data is the first and perhaps most critical step in the ISA methodology. Once the problem or application domain is specified, a large collection of test instances must be curated by gathering together existing benchmark test suites, that are often scattered across different websites and data repositories. MATILDA's website [43] provides a library of benchmark instances for various optimization problems (e.g., 0-1 Knapsack, Graph Coloring, Traveling Salesman Problem), as well as common learning and model fitting problems (e.g., Anomaly Detection, Classification, Regression). Alternatively, numerous research communities have developed their own libraries of benchmark instances. For example, OpenML [45] provides large collections of classification and regression datasets; BBOB [14] is one of the standard benchmark suites for continuous black-box optimization. The next step is to select a set of complementary, state-of-the-art algorithms, with the libraries mentioned above including experimental results for most of them. If the goal is to use ISA to report the strengths and weaknesses of a newly proposed algorithm in a publication, it will be necessary to ensure state-of-the-art algorithms are included in the analysis for comparison.

Once the instances have been collected, it is necessary to identify some meaningful features that capture their intrinsic difficulties, and calculate their feature vectors. This often requires the design of specialized methods, for which significant domain knowledge is essential. That is, an understanding of the relevant characteristics found in an instance, the different algorithms available to solve such problems, and how those algorithms are affected by those characteristics. In the last few decades, there have been significant advances on the development of features for different problem domains, e.g. fitness landscape analysis metrics for black-box optimization [21], complexity measures for machine learning from the field of meta-learning [4], features for time series data [46], and numerous difficulty metrics for combinatorial optimization problems [42].

Broadly speaking, the ISA toolkit [32] expects the meta-data to be stored in a Comma-Separated Values (CSV) file format, named `'metadata'`, where each row represents a unique instance of a problem, and each column represents either: (a) a unique identifier for the instance; (b) the instance class family or source of the instance; (c) a feature, which represents a measurable mathematical or statistical property of the instance; or (d) a performance metric of a given algorithm when run on the instance. Figure 2 illustrates the header of the CSV file for the University Curriculum Timetabling problem meta-data available in MATILDA, where the instance and source identifiers are of type string recorded in the columns `'Instances'` and `'Source'` respectively. Each feature and algorithm are numerical types identified by their name preceded by the string `'feature_'` or `'algo_'` respectively. The columns can be ordered in any sequence as long as the identifiers are correctly used. Moreover, while empty cells, NaN or null values are allowed, they are not recommended and should only be a small minority in the data. Common mistakes in formatting the data that result in incorrect phrasing are: to use `'NA'` instead of `'NaN'` to identify null values, to have Excel error codes such as `'#REF!'`, `'#NULL!'`, or `'#DIV/0!'`, or leaving rows empty.

3.2 Constructing the Instance Space

The construction of an instance space is performed by 4 methods, known as PRELIM, SIFTED, PILOT and CLOISTER, which are executed sequentially by the `‘buildIS’` script. Their tasks are:

(1) PRELIM: Preparation for Learning of Instance Meta-data. This method prepares the meta-data by specifying a binary measure of "good" performance, and bounding and scaling the meta-data for subsequent machine learning.
(2) SIFTED: Selection of Instance Features to Explain Difficulty. This method selects a subset of relevant features by considering correlations with algorithm performance and eliminating redundancies.
(3) PILOT: Projecting Instances with Linearly Observable Trends. This method is a novel dimension reduction technique that projects instances from a high dimensional feature space to the $2D$ instance space in a manner that encourages linear trends in features and algorithm performance distributions to support visualizations.
(4) CLOISTER: Correlated Limits of the Instance Space's Theoretical or Experimental Regions. This method approximates the boundary of the instance space using experimental and/or theoretical upper and lower bounds of the instance features, adjusting for correlations between features.

Implementation details of each of these four methods are provided in the following subsections, including the pseudocode for the five algorithms used to create the instance space.

3.2.1 PRELIM: Preparation for Learning of Instance Meta-data. Assuming that the meta-data has been collected and presented in the correct structure, the ISA toolkit pre-processes the meta-data using PRELIM, such that it becomes amenable to the learning and visualization methods used in the subsequent stages. Initially, PRELIM calculates a binary measurement of "good" performance from the data based on user defined specifications. This is a somewhat arbitrary definition, but exerts significant influence on the final results. Good performance can be defined by the user in two ways: (a) *absolutely*, where an algorithm's performance is deemed good if its performance metric exceeds a threshold value, e.g., classification accuracy for a model should be above 80%; or (b) *relatively*, where an algorithm's performance is deemed good if its performance metric is within a margin from the best of the other algorithms in $\mathcal{A}$, e.g., classification accuracy should be within 5% of the best algorithm's accuracy.

Once good performance has been defined, each feature is bounded between its median plus or minus five times its interquartile range (IQR) to reduce the effect of outliers. In other words, any value that exceeds this range is set to the bounds. We use the median and the IQR as they are robust estimators of the typical value and the spread. Next, PRELIM applies the one parameter Box-Cox transformation to each feature and performance measure to stabilize their variance and normalize the data. The transformation is defined by the equation:

$$\breve{\mathrm{f}}_i = \begin{cases} \frac{\mathrm{f}_i^{\lambda}-1}{\lambda} & \text{if } \lambda \neq 0 \\ \ln(\mathrm{f}_i) & \text{if } \lambda = 0 \end{cases} \tag{1}$$

where f_i must be a value greater than zero. As such, before the transformation, the values are shifted by adding one minus the minimum to each feature or the machine precision value ε to each performance measure. The value of the Box-Cox transformation parameter λ is estimated by maximizing the Log-Likelihood function. Finally, the toolkit applies a z-transform to standardize each feature and performance value to ensure a mean of zero and standard deviation of one. Algorithm 1 describes the steps followed to augment the meta-data to include a binary "good" (1) or "not good" (0) performance label for each instance and algorithm pair, and to bound and scale the meta-data to complete the pre-processing tasks.

3.2.2 SIFTED: Selection of Instance Features to Explain Difficulty. After pre-processing the meta-data, the toolkit uses SIFTED to perform automated feature subset selection, identifying those features that are uncorrelated

Algorithm 1: Preparation for Learning of Instance Meta-data (PRELIM) method

Input: A matrix $\mathbf{F} \in \mathbb{R}^{m \times n}$ of instance features, a matrix $\mathbf{Y} \in \mathbb{R}^{a \times n}$ of performance measures, a performance threshold, ϵ, a binary flag ϕ_{max} defining minimization or maximization, a binary flag ϕ_{abs} defining absolute or relative performance, and the binary flags $\{\phi_{\text{bnd}}, \phi_{\text{nrm}}\}$ defining whether bounding and normalization are performed.

Output: A matrix $\mathbf{F} \in \mathbb{R}^{m \times n}$ of instance features, a matrix $\mathbf{Y} \in \mathbb{R}^{a \times n}$ of performance measures, a matrix $\mathbf{Y}_{\text{bin}} \in \mathbb{B}^{a \times n}$ of binary performance measures, a vector $\mathbf{y}^* \in \mathbb{R}^n$ of best performance values, a vector $\mathbf{p} \in \{1, \ldots, a\}^n$ of best performing algorithms, and a set of output parameters, Π.

```
Function PRELIM ($\mathbf{F}, \mathbf{Y}, \epsilon, \phi_{max}, \phi_{abs}, \phi_{bnd}, \phi_{nrm}$) is
    // Calculation of the binary measure of "good" performance
    if $\phi_{max} = TRUE$ then
        $\mathbf{Y}[\mathbf{Y} = \text{NaN}] \leftarrow -\infty$;
        $\{\mathbf{y}^*, \mathbf{p}\} \leftarrow$ FindMaxByCol ($\mathbf{Y}$);    // Returns the minimum and its index for each column
        if $\phi_{abs} = TRUE$ then $\mathbf{Y}_{\text{bin}} \leftarrow \mathbf{Y} \geq \epsilon$;
        else
            $\mathbf{Y} \leftarrow 1 - (\mathbf{Y} \oslash$ RowVectorAsMatrix $(\mathbf{y}^*, a))$;    // Apply an element-wise division
            $\mathbf{Y}_{\text{bin}} \leftarrow \mathbf{Y} \leq \epsilon$;
        end
    else
        $\mathbf{Y}[\mathbf{Y} = \text{NaN}] \leftarrow \infty$;
        $\{\mathbf{y}^*, \mathbf{p}\} \leftarrow$ FindMinByCol ($\mathbf{Y}$);    // Returns the maximum and its index for each column
        if $\phi_{abs} = FALSE$ then
            $\mathbf{Y} \leftarrow (\mathbf{Y} \oslash$ RowVectorAsMatrix $(\mathbf{y}^*, a)) - 1$;    // Apply an element-wise division
        end
        $\mathbf{Y}_{\text{bin}} \leftarrow \mathbf{Y} \leq \epsilon$;
    end
    // Pre-processing function for bounding and scaling of the meta-data
    if $\phi_{bnd} = TRUE$ then
        $\Pi.\mathbf{f}_{\text{med}} \leftarrow$ FindMedianByRow ($\mathbf{F}$);
        $\Pi.\boldsymbol{\iota} \leftarrow$ FindIQRByRow ($\mathbf{F}$);
        $\mathbf{F}_{\text{max}} \leftarrow$ ColVectorAsMatrix ($\Pi.\mathbf{f}_{\text{med}} + 5\Pi.\boldsymbol{\iota}, n$);
        $\mathbf{F}_{\text{min}} \leftarrow$ ColVectorAsMatrix ($\Pi.\mathbf{f}_{\text{med}} - 5\Pi.\boldsymbol{\iota}, n$);
        $\mathbf{F} \leftarrow \mathbf{F} \circ (\mathbf{F} \geq \mathbf{F}_{\text{min}} \| \mathbf{F} \leq \mathbf{F}_{\text{max}}) + \mathbf{F}_{\text{min}} \circ (\mathbf{F} \leq \mathbf{F}_{\text{min}}) + \mathbf{F}_{\text{max}} \circ (\mathbf{F} \geq \mathbf{F}_{\text{max}})$;
    end
    if $\phi_{nrm} = TRUE$ then
        $\Pi.\mathbf{f}_{\text{min}} \leftarrow$ FindMinByRow ($\mathbf{F}$);
        $\mathbf{F} \leftarrow \mathbf{F} -$ ColVectorAsMatrix ($\Pi.\mathbf{f}_{\text{min}} + 1$);
        $\{\mathbf{F}, \Pi.\boldsymbol{\lambda}_{\mathbf{f}}\} \leftarrow$ BoxCoxByRow ($\mathbf{F}$);
        $\{\mathbf{F}, \Pi.\boldsymbol{\mu}_{\mathbf{F}}, \Pi.\boldsymbol{\sigma}_{\mathbf{F}}\} \leftarrow$ ZScoreByRow ($\mathbf{F}$);
        $\mathbf{Y}[\mathbf{Y} = 0] \leftarrow \varepsilon$;
        $\{\mathbf{Y}, \Pi.\boldsymbol{\lambda}_{\mathbf{Y}}\} \leftarrow$ BoxCoxByRow ($\mathbf{Y}$);
        $\{\mathbf{Y}, \Pi.\boldsymbol{\mu}_{\mathbf{Y}}, \Pi.\boldsymbol{\sigma}_{\mathbf{Y}}\} \leftarrow$ ZScoreByRow ($\mathbf{Y}$);
    end
    return $\{\mathbf{F}, \mathbf{Y}, \mathbf{Y}_{\text{bin}}, \mathbf{y}^*, \mathbf{p}, \Pi\}$;
end
```

with each other and correlated most strongly with algorithm performance. The aim is to find a subset of the candidate features in the meta-data set $\mathcal{F}$ that best explain the difficulty of the test instances in $\mathbf{I}$. This is achieved by SIFTED in a two-step process, as described in Algorithm 2. First, SIFTED calculates the absolute value of the Pearson correlation between the features and the algorithm performance. Then, it selects the most correlated

Algorithm 2: Selection of Instance Features to Explain Difficulty (SIFTED)

```
Input: A matrix $\mathbf{F} \in \mathbb{R}^{m \times n}$ of instance features, a matrix $\mathbf{Y} \in \mathbb{R}^{a \times n}$ of performance measures, a matrix $\mathbf{Y}_{\text{bin}} \in \mathbb{B}^{a \times n}$ of binary
       performance measures, and a desired number of features, $K$.
Output: A matrix $\tilde{\mathbf{F}} \in \mathbb{R}^{q \times n}$ of instance features, and a set of output parameters, $\Pi$.
1  Function SIFTED $(\mathbf{F}, \mathbf{Y}, \mathbf{Y}_{bin}, K)$ is
2      $\{\Pi.\mathbf{R}_{\mathbf{F},\mathbf{Y}}, \Pi.\mathbf{P}_{\mathbf{F},\mathbf{Y}}\} \leftarrow$ PearsonCorrByRow $(\mathbf{F}, \mathbf{Y})$;
3      $\mathbf{R} \leftarrow \Pi.\mathbf{R}_{\mathbf{F},\mathbf{Y}}$;
4      $\mathbf{R}[\mathbf{R} = \text{NaN} \parallel \Pi.\mathbf{P}_{\mathbf{F},\mathbf{Y}} > 0.05] \leftarrow 0$;
5      $\text{idx}_1 \leftarrow$ FindIdxMaxByRow $(|\mathbf{R}|)$;            // Index of the maximum absolute value for each row
6      $\text{idx}_2 \leftarrow$ FindIdxByRow $(|\mathbf{R}| \geq 0.3)$;     // Index of the elements with absolute value greater than 0.3
7      $\Pi.\text{idx}_{\text{cor}} \leftarrow$ FindUniqueValues $(\{\text{idx}_1, \text{idx}_2\})$;
8      $\tilde{\mathbf{F}} \leftarrow \mathbf{F}[\Pi.\text{idx}_{\text{cor}}, :]$;
9      $\Pi.\mathbf{R}_{\mathbf{F},\mathbf{F}} \leftarrow$ PearsonCorrByRow $(\tilde{\mathbf{F}}, \tilde{\mathbf{F}})$;
10     $\Pi.\mathbf{C} \leftarrow$ KMeans $(1 - |\Pi.\mathbf{R}_{\mathbf{F},\mathbf{F}}|, K)$;
11     $\Pi.\text{idx}_{\text{clu}} \leftarrow$ FindOptimalCombination $(\Pi.\mathbf{C}, \tilde{\mathbf{F}}, \mathbf{Y}_{\text{bin}})$;
12     $\tilde{\mathbf{F}} \leftarrow \tilde{\mathbf{F}}[\Pi.\text{idx}_{\text{clu}}, :]$;
13     return $\{\tilde{\mathbf{F}}, \Pi\}$;
14 end
```

feature per algorithm, and any other feature whose correlation is at least moderate, i.e., above 0.3 [15], with at least one algorithm.

Next, the toolkit identifies groups of similar features using the k-means clustering algorithm with a dissimilarity measure of $1 - |\rho_{i,j}|$, where $\rho_{i,j}$ is the correlation between two features. The number of clusters is a user defined value, with a minimum of 3 and a default of 10. Then, taking one feature from each cluster, the toolkit determines which combination has the lowest predictive error when projected into a temporary $2D$ space (which is not the final instance space). This is achieved by taking a candidate feature subset, one feature from each cluster, and projecting into $2D$ using Principal Component Analysis (PCA). The resulting coordinates become the inputs to a set of Random Forest (RF) models, each one of them predicting whether an algorithm $\alpha \in \mathcal{A}$ is "good" according to the binary measure specified by Algorithm 1. The average out-of-the-bag error estimate given by the RF models is used as the loss function. Both PCA and RFs are used at this stage, as they provide computationally cheaper alternatives to the algorithms used in later stages. In particular, PCA is a proven suboptimal solution to the underlying optimization problem that our dimensional reduction method PILOT solves [30], which we describe in the following section.

3.2.3 PILOT: Projecting Instances with Linearly Observable Trends. Now that a valuable subset of instance features has been identified, the toolkit can construct an optimal $2D$ instance space that aims to facilitate the identification of relationships between instance features and the performance of algorithms. In practice, this means creating a coordinate system that systematically locates the instances in the $2D$ plane to facilitate visualization of trends in relationships across the instance space: directions of hardness increasing from one edge of the space to the opposite. In addition to creating linear trends in algorithm performance across the $2D$ plane, we also seek linear trends in the feature distributions so that explanations of algorithm performance can be deduced in terms of the instance features. To achieve a projection that encourages linear trends in performance and feature distributions, we have developed a customized dimension reduction method using optimization to achieve the goal of linear trends, rather than employing other dimension reduction methods such as PCA that have different goals (e.g. maximizing retained variance). The previous method SIFTED has already increased the chances of observing trends across the instance space by selecting the features with the most significant explanatory power. The way

Algorithm 3: Projecting Instances with Linearly Observable Trends (PILOT)

```
Input: A matrix $\tilde{F} \in \mathbb{R}^{q \times n}$ of instance features, a matrix $Y \in \mathbb{R}^{a \times n}$ of performance measures, and a number of random restarts $N_{\text{try}}$,
    and a binary flag $\phi_{\text{num}}$ that determines whether the analytical or the numerical solution is calculated.
Output: A matrix $Z \in \mathbb{R}^{n \times 2}$ of coordinates in the $2D$ instance space, and a set of projection matrices $\{A_r, B_r, C_r\}$.
1  Function PILOT $\left(\tilde{F}, Y, N_{try}, \phi_{num}\right)$ is
2      if $\phi_{num} = FALSE$ then
3          $\overline{X} \leftarrow \left[\tilde{F}; Y\right]$;
4          $V \leftarrow$ FindTopEigenVectors $\left(\overline{X}\,\overline{X}^{\top}, 2\right)$;
5          $B_r \leftarrow V[1:n,:]$;
6          $C_r \leftarrow V[n+1:m,:]$;
7          $X_r \leftarrow \tilde{F}^{\top}\left(\tilde{F}\tilde{F}^{\top}\right)^{-1}$;
8          $A_r \leftarrow V^{\top}\overline{X}X_r$;
9          $Z \leftarrow A_i\tilde{F}$;
10     else
11         $D_H \leftarrow$ EuclideanDist $\left(\tilde{F}\right)$;                // Distance between instances in the feature space
12         $D_H \leftarrow$ MatrixAsRowVector $(D_H)$;                          // Reshape as a column vector
13         $\rho_{\text{best}} \leftarrow -\infty$;
14         for $i = 1$ to $N_{try}$ do                                          // Repeat $N_{\text{try}}$ times
               // Initialise the projection matrices randomly between $[-1, 1]$
15             $A_0 \leftarrow$ UniformRandMatrix $(2, m, [-1, 1])$;
16             $B_0 \leftarrow$ UniformRandMatrix $(m, 2, [-1, 1])$;
17             $C_0 \leftarrow$ UniformRandMatrix $(a, 2, [-1, 1])$;
18             $\{A_i, B_i, C_i\} \leftarrow$ BFGS $(\mathcal{D}, A_0, B_0, C_0)$;       // Use BFGS to find a solution to $\mathcal{D}$
19             $Z_i \leftarrow A_i\tilde{F}$;
20             $D_L \leftarrow$ EuclideanDist $(Z_i)$;
21             $D_L \leftarrow$ MatrixAsRowVector $(D_L)$;
22             $\rho_i \leftarrow$ PearsonCorrByRow $(D_H, D_L)$;
23             if $\rho_{best} < \rho_i$ then $\{A_r, B_r, C_r, Z, \rho_{\text{best}}\} \leftarrow \{A_i, B_i, C_i, Z_i, \rho_i\}$;   // Best solution so far
24         end
25     end
26     return $\{Z, A_r, B_r, C_r\}$;
27 end
```

these features are combined to construct a new $2D$ coordinate system defining the instance space is described by the Projecting Instances with Linearly Observable Trends (PILOT) method[2], which was introduced in previous work [30] and whose details we present below for reference. Algorithm 3 describes its operation in pseudocode.

An ideal projection of the instances is one that creates linear trends — with low values at one end and high values at the other end of a straight line — when each algorithm's performance metric, and each feature value, is inspected for every instance. In other words, we would like a well-fit linear model for each feature and each the performance metric of each algorithm, based on the instance location in the $2D$ plane. Mathematically, this involves finding the matrices $A_r \in \mathbb{R}^{2\times q}$, $B_r \in \mathbb{R}^{q\times 2}$ and $C_r \in \mathbb{R}^{a\times 2}$ which minimize the total approximation error of the $q + a$ linear models:

$$||\tilde{F} - \widehat{F}||_F^2 + ||Y - \widehat{Y}||_F^2 \tag{2}$$

[2]In previous work we referred to this method as Prediction Based Linear Dimensionality Reduction (PBLDR).

such that:

$$
\begin{aligned}
\mathbf{Z} &= \mathbf{A}_r\tilde{\mathbf{F}} && (3)\\
\widehat{\mathbf{F}} &= \mathbf{B}_r\mathbf{Z} && (4)\\
\widehat{\mathbf{Y}} &= \mathbf{C}_r\mathbf{Z} && (5)
\end{aligned}
$$

where $\mathbf{Z} \in \mathbb{R}^{n\times 2}$ is the matrix of instance coordinates in the $2D$ space. We assume that $q < n$ and $\tilde{\mathbf{F}}$ is full row rank, i.e. rank $\left(\tilde{\mathbf{F}}\right) = q$. If $\tilde{\mathbf{F}}$ is not full rank, then we consider the problem in a subspace spanned by $\tilde{\mathbf{F}}$. Thus, PILOT solves the following optimization problem $(\mathcal{D})$:

$$
(\mathcal{D}) \qquad
\begin{array}{ll}
\min & \left\|\tilde{\mathbf{F}} - \mathbf{B}_r\mathbf{Z}\right\|_F^2 + \|\mathbf{Y} - \mathbf{C}_r\mathbf{Z}\|_F^2 \\
\text{s.t.} & \mathbf{Z} = \mathbf{A}_r\tilde{\mathbf{F}} \\
 & \mathbf{A}_r \in \mathbb{R}^{2\times q} \\
 & \mathbf{B}_r \in \mathbb{R}^{q\times 2} \\
 & \mathbf{C}_r \in \mathbb{R}^{a\times 2}
\end{array}
\qquad (6)
$$

In previous work [30] we proved the existence of a global optimum for $\mathcal{D}$ with infinitely many solutions. A lower bound for $\mathcal{D}$ is given by the two largest principal components of the matrix $\overline{\mathbf{F}} = \left(\tilde{\mathbf{F}}^\top\mathbf{Y}\right)^\top$, which would correspond to the solution if $\tilde{\mathbf{F}}\tilde{\mathbf{F}}^\top$ were invertible. However, this is not often the case and this solution is numerically unstable. Therefore, PILOT numerically solves $(\mathcal{D})$, which is known to be convex but highly ill-conditioned with an infinite number of solutions falling within a line, using the Broyden–Fletcher–Goldfarb–Shanno (BFGS) optimization algorithm [5]. We represent $\{\mathbf{A}_r, \mathbf{B}_r, \mathbf{C}_r\}$ as a column vector by concatenating the matrices' columns. BFGS always finds a global optimum for $\mathcal{D}$; therefore, the best solution is the one with the highest topological preservation, defined as the Pearson Correlation between the distances in the feature space, $\left\|\mathbf{f}_i - \mathbf{f}_j\right\|$, and the distances in the instance space, $\left\|\mathbf{z}_i - \mathbf{z}_j\right\|$ [48], from a number of repeats N_{try}, which is set to 30 by default.

The solution of $\mathcal{D}$ by the PILOT method now provides the linear transformation matrix $\mathbf{A}_r$ needed to map each instance from its m-dimensional feature vector to its location in the $2D$ instance space $\mathbf{Z}$ given by Equation (6).

3.2.4 CLOISTER: Correlated Limits of the Instance Space's Theoretical or Experimental Regions. The instance space now represents the available instances as points in a $2D$ space, with the convex hull of the point cloud representing the area in which there is empirical evidence that instances do exist. However, we can define an expanded boundary if we consider all the feasible combinations of the features, given their theoretical or empirical upper and lower bounds. Knowing this boundary would give us information about the diversity of the existing problem subset, $\mathbf{I}$ within the realm of theoretically possible instances $\mathcal{P}$ of the problem. For this purpose, we estimate this mathematical boundary using the Correlated Limits of the Instance Space's Theoretical or Experimental Regions (CLOISTER) method, whose pseudocode is presented in Algorithm 4.

Let $\mathbf{R} \in \mathbb{R}^{q\times q}$ be the matrix of correlations between features. We define $\mathbf{f}_{\text{U}} = \left[\text{f}_{\text{U},1} \ \dots \ \text{f}_{\text{U},q}\right]^\top$ and $\mathbf{f}_{\text{L}} = \left[\text{f}_{\text{L},1} \ \dots \ \text{f}_{\text{L},q}\right]^\top$ to be vectors containing the upper and lower bounds from the rows of $\tilde{\mathbf{F}}$. A vertex vector, $\mathbf{v}_i$, is defined as a combination of values from $\mathbf{f}_{\text{U}}$ and $\mathbf{f}_{\text{L}}$, such that only the upper or lower bound of a feature is included, e.g., $\mathbf{v}_1 = \left[\text{f}_{\text{U},1}\ \text{f}_{\text{L},2} \ \dots \ \text{f}_{\text{L},q}\right]^\top$, which represents a state whereby feature 1 is at its minimum value, and all other features are at their maximum. We define $\mathbf{V} = [\mathbf{v}_1 \ \dots \ \mathbf{v}_c] \in \mathbb{R}^{q\times c}$, $c = 2^q$ to be the matrix containing all possible vertex vectors, which defines a hyper-cube that encloses all the instances in $\mathbf{I}$.

Algorithm 4: Correlated Limits of the Instance Space's Theoretical or Experimental Regions (CLOISTER)

```
Input: A matrix $\tilde{\mathbf{F}} \in \mathbb{R}^{m\times n}$ of instance features, a minimum correlation value for which a par of features would be considered
       co-linear, $\epsilon$, and a $p$-value for which a pair of features would be considered uncorrelated, $p$.
Output: A matrix $\mathbf{Z}_{\text{edge}} \in \mathbb{R}^{n\times 2}$ of boundary coordinates in the $2D$ instance space.
1  Function CLOISTER $\left(\tilde{\mathbf{F}}, \epsilon, p\right)$ is
2      $\left\{\rho_{\mathrm{F,F}}, p_{\mathrm{F,F}}\right\} \leftarrow$ PearsonCorrByRow $\left(\tilde{\mathbf{F}}, \tilde{\mathbf{F}}\right)$;
3      $\rho_{\mathrm{F,F}}\left[p_{\mathrm{F,F}} \geq p\right] \leftarrow 0$;
4      $\mathbf{f}_L \leftarrow$ FindMinByRow $\left(\tilde{\mathbf{F}}\right)$;
5      $\mathbf{f}_U \leftarrow$ FindMaxByRow $\left(\tilde{\mathbf{F}}\right)$;
6      $\mathbf{V} \leftarrow$ FindAllCombinations $(\mathbf{f}_L, \mathbf{f}_U)$;        // Generate a hyper-cube enclosing all instances
7      $c \leftarrow$ NumberOfColumns $(\mathbf{V})$;
8      for $i = 1$ to $c$ do
9          for $j = 1$ to $m$ do
10             for $i = j + 1$ to $m$ do
11                 if $\rho_{\mathrm{F,F}}[j,k] > \epsilon$ && Sign $(\mathbf{V}[j,i]) \neq$ Sign $(\mathbf{V}[k,i])$ then $\mathbf{r}_i$ = TRUE;
12                 else if $\rho_{\mathrm{F,F}}[j,k] < -\epsilon$ && Sign $(\mathbf{V}[j,i]) =$ Sign $(\mathbf{V}[k,i])$ then $\mathbf{r}_i$ = TRUE;
13             end
14         end
15     end
16     $Z_c \leftarrow \mathbf{A}_i \mathbf{V}[\neg\mathbf{r}, :]$;
17     $\mathbf{Z}_{\text{edge}} \leftarrow$ FindConvexHull $(\mathbf{Z}_c)$;
18     return $\left\{\mathbf{Z}_{\text{edge}}\right\}$;
19 end
```

Now, some of the vectors in $\mathbf{V}$ may represent combinations of features that are unlikely to coexist due to strong correlations between features limiting their range of combined values. For example, if feature 1 and feature 2 are strongly positively correlated, we are unlikely to ever find instances that have a high value of feature 1 and a low value of feature 2, making the vertex vector $\mathbf{v}_1 = \left[\mathrm{f}_{\mathrm{U},1}\ \mathrm{f}_{\mathrm{L},2}\ \ldots\ \mathrm{f}_{\mathrm{L},q}\right]^\top$ unlikely to be near any true instances. To prune back the vertex vectors to only those likely to define the boundary of the instances, we compare the correlation, $\rho_{i,j}$, of any two features $\mathbf{f}_i$ and $\mathbf{f}_j$. Given a user-defined threshold ϵ, a vertex vector cannot simultaneously contain $\left\{\mathrm{f}_{\mathrm{L},i}, \mathrm{f}_{\mathrm{U},j}\right\}$ or $\left\{\mathrm{f}_{\mathrm{U},i}, \mathrm{f}_{\mathrm{L},j}\right\}$ if $\rho_{i,j} > \epsilon$. Also, the vertex vector cannot simultaneously contain $\left\{\mathrm{f}_{\mathrm{U},i}, \mathrm{f}_{\mathrm{U},j}\right\}$ or $\left\{\mathrm{f}_{\mathrm{L},i}, \mathrm{f}_{\mathrm{L},j}\right\}$ if $\rho_{i,j} < -\epsilon$. After eliminating such unlikely vertex vectors, the edges connecting the remaining vertex vectors are projected into the two-dimensional instance space $2D$ using $\mathbf{A}_r$ from Equation (6), resulting in a matrix $\mathbf{Z}_c$, whose convex hull now represents the mathematical boundary of the instance space. We assume in Algorithm 4 that the upper and lower bounds of features are based on experimental observation from the instances' features $\tilde{\mathbf{F}}$, however in some problem domains much is known theoretically about the bounds of features, e.g. graph density is bounded by $[0, 1]$, and these theoretical bounds can substituted where known.

The methods known as PRELIM, SIFTED, PILOT and CLOISTER, implemented by Algorithms 1 to 4, have now generated a $2D$ instance space containing all collected test instances in the available meta-data, and a projection of their mathematical boundary. An analysis of algorithm performance across this instance space, using visualizations and machine learning methods, will next provide predictions about how algorithms are likely to perform across the set of all possible instances $\mathcal{P}$, and whether additional test instances are needed to understand how algorithms will perform across this broad instance space.

Algorithm 5: Performance prediction and automated algorithm selection (PYTHIA)

Input: A matrix $\mathbf{Z} \in \mathbb{R}^{n\times 2}$ of coordinates in the $2D$ instance space, a matrix $\mathbf{Y} \in \mathbb{R}^{a\times n}$ of performance measures, a matrix of binary performance measures, $\mathbf{Y}_{\text{bin}} \in \mathbb{B}^{a\times n}$, a vector of best performance values, $\mathbf{y}^* \in \mathbb{R}^n$, and a vector of best performing algorithms, $\mathbf{p} \in \{1,\ldots,a\}^n$.

Output: A set of SVM models $\mathcal{S}$, a matrix of cross-validated estimated binary performance measures, $\widehat{\mathbf{Y}}_{\text{cv,bin}} \in \mathbb{B}^{a\times n}$, a vector of cross-validated estimated best performing algorithms, $\hat{\mathbf{p}}_{\text{cv}} \in \{1,\ldots,a\}^n$, and its equivalents using the full dataset as training data, $\widehat{\mathbf{Y}}_{\text{bin}}$ and $\hat{\mathbf{p}}$, a tensor $\mathbf{C} \in \mathbb{B}^{k,n,a}$ that identifies the cross-validation sets.

```
1  Function PYTHIA ($\mathbf{Z}, \mathbf{Y}, \mathbf{Y}_{bin}, \mathbf{y}^*, \mathbf{p}$) is
2      $\{\mathbf{Z}_{\text{norm}}, \Pi.\boldsymbol{\mu}_{\mathbf{Z}}, \Pi.\boldsymbol{\sigma}_{\mathbf{Z}}\} \leftarrow$ ZScoreByRow ($Z$)
3      for $i = 1$ to $a$ do
4          $\mathbf{C}[:,:,i] \leftarrow$ CreateCVPartition ($\mathbf{Y}_{\text{bin}}[i,:], k$);
5          $\{\mathcal{S}[i], \widehat{\mathbf{Y}}_{\text{cv,bin}}, \widehat{\mathbf{Y}}_{\text{bin}}, \Pi.C[i], \Pi.\gamma[i]\} \leftarrow$ CrossValidatedSVMTrain ($\mathbf{Z}_{\text{norm}}, \mathbf{Y}_{\text{bin}}[i,:], \mathbf{C}[:,:,i]$);
6          $\{\Pi.P[i], \Pi.R[i], \Pi.A[i]\} \leftarrow$ CalculateSVMPerformance ($\mathbf{Y}_{\text{bin}}, \widehat{\mathbf{Y}}_{\text{cv,bin}}$);
7      end
8      $\hat{\mathbf{p}}_{\text{cv}} \leftarrow$ FindMaxIndexByCol ($\widehat{\mathbf{Y}}_{\text{cv,bin}} \circ$ ColVectorAsMatrix ($\Pi.P, n$));
9      $\hat{\mathbf{p}} \leftarrow$ FindMaxIndexByCol ($\widehat{\mathbf{Y}}_{\text{bin}} \circ$ ColVectorAsMatrix ($\Pi.P, n$));
10     return $\{\mathcal{S}, \widehat{\mathbf{Y}}_{\text{cv,bin}}, \widehat{\mathbf{Y}}_{\text{bin}}, \hat{\mathbf{p}}_{\text{cv}}, \hat{\mathbf{p}}, \Pi\}$
11 end
```

3.3 Automated Algorithm Selection

One advantage of the instance space is that it can be partitioned into regions of strength or weakness for each algorithm, and this information can generate automated algorithm recommendations for untested instances using their known coordinates $\mathbf{Z}$ based on features. For this purpose, we have developed PYTHIA, whose pseudocode is presented in Algorithm 5. PYTHIA trains a Support Vector Machine (SVM) for each algorithm, using as inputs the normalized coordinates $\mathbf{Z}$ obtained from PILOT, and generates as output the predicted binary measure of performance described in Section 3.2.1. PYTHIA can use either the MATLAB-native or the LIBSVM [7] implementation of the SVM for this partitioning, with polynomial or Gaussian kernels. Moreover, PYTHIA self-tunes the SVM parameters $\{C, \gamma\}$ depending on the implementation. For the MATLAB implementation, self-tuning is achieved using 30 iterations of the Bayesian Optimization algorithm bounded between $\left[2^{-10}, 2^4\right]$, with k-fold stratified cross-validation (CV). The probability of improvement loss function, is defined as:

$$PI(x) = \Phi\left(\frac{\mu_Q(x_{\text{best}}) - m - \mu_Q(x)}{\sigma_Q(x)}\right) \tag{7}$$

where x is a new parameter estimate vector, x_{best} is the location of the lowest posterior mean, $\mu_Q(x_{\text{best}})$ is the lowest value of the posterior mean, $\sigma_Q(x)$ is the posterior standard deviation at x, and $\Phi(\cdot)$ is the unit normal cumulative distribution function. On the other hand, for the LIBSVM implementation, self-tuning is achieved using 30 iterations of the random search algorithm, using a Latin Hyper-cube design bounded between $\left[2^{-10}, 2^4\right]$ as sample points, with k-fold stratified CV, and using model error as the loss function. Once the best hyperparameters are identified and the CV results are collected, the SVMs are trained with the complete training dataset of all instances $\mathbf{I}$.

A suitable algorithm is selected by providing the coordinates $\mathbf{Z}$ of a new instance to the SVMs. Once predicted, ties are resolved by recommending the algorithm whose model has the highest precision. If no algorithm is deemed good by any SVM, PYTHIA recommends by default the algorithm with the highest average performance,

Algorithm 6: Good and Best Footprints via the Triangulation with Removal of Areas with Contradicting Evidence (TRACE) method

```
Input: A matrix $\mathbf{Z} \in \mathbb{R}^{n\times 2}$ of coordinates in the $2D$ instance space, a matrix $\mathbf{Y} \in \mathbb{B}^{n\times a}$ of binary performance measures, and a vector
       $\mathbf{p} \in \{1,\ldots,a\}^{n\times 1}$ of indexes indicating the best performing algorithm for an instance.
Output: Two footprint sets, $\{\Phi_{\text{good}}, \Phi_{\text{best}}\}$, which correspond to good and best performance.
Function FootprintAnalysis($\mathbf{Z}, \mathbf{Y}, \mathbf{p}$) is
    $S \leftarrow$ TRACEbuild $(\mathbf{Z}, \mathbf{1})$;          // Calculate the area, density and purity of the space
    for $i = 1$ to $a$ do                                 // Build two footprints for all algorithms
        $\Phi_{\text{good},i} \leftarrow$ TRACEbuild $(\mathbf{Z}, \mathbf{y}_i)$;
        $\Phi_{\text{best},i} \leftarrow$ TRACEbuild $(\mathbf{Z}, \mathbf{p} = i)$;
    end
    for $i = 1$ to $a$ do
        for $j = i + 1$ to $a$ do
            // Compare the best performance footprints for two different algorithms and retain the areas
               with the highest confidence
            $\{\Phi_{\text{best},i}, \Phi_{\text{best},j}\} \leftarrow$ TRACEcomp $(\Phi_{\text{best},i}, \Phi_{\text{best},j}, \mathbf{Z}, \mathbf{p} = i, \mathbf{p} = j)$;
        end
    end
end
```

while marking the instance as not having a strong algorithm. The accuracy of the SVMs is highly dependent on the quality of the meta-data and the binary performance metric produced in Section 3.2.1.

While machine learning prediction models are useful for automated algorithm selection, it is possible that interesting classes of instances eliciting unique algorithm behaviors may be lost as the SVM attempts to maximize average accuracy across all instances. It is for this reason that we also rely on the visualization offered by the instance space, rather than machine learning models alone, to gain additional insights into algorithm strengths and weaknesses for different types of instances. Algorithm footprints, as described in the following section, are a core component of visualizing and quantifying a more nuanced assessment of algorithm performance.

3.4 Generating Algorithm Footprints and Metrics

An algorithm's *footprint* is the generalized area of the instance space where good performance (however defined by the user), or best performance, is expected based on inference from empirically observed performance data [44]. Algorithm footprints are a key concept of ISA, as they provide a more objective approach to rigorous assessment of algorithm performance that simultaneously considers instance characteristics, along with the strength of evidence, the diversity and possible bias of the instances, and the existence of any contradictory information that may weaken the conclusions. Besides its area, α, a footprint is characterized by its density, d, defined as the number of instances enclosed by the footprint per unit area, and its purity, p defined as the percentage of all instances for which the algorithm has good performance that are enclosed by the footprint.

The procedure to construct a footprint has been refined from its initial version [44] to reduce parameters, improve repeatability, robustness and stability. The final method for footprint identification is known as Triangulation with Removal of Areas with Contradicting Evidence (TRACE), with Algorithms 6 to 8 describing its details. TRACE follows three core steps: (a) estimating the area and density, α_S and d_S respectively, of the convex hull containing all instances in $\mathbf{I}$, which are used as baseline metrics to normalize each algorithm's footprint as a percentage of the instance space; (b) building and characterizing the good and best footprints for each algorithm; and (c) comparing the best footprints to reduce or eliminate overlapping sections with weak evidence.

Algorithm 7: TRACEbuild: Footprint construction algorithm

```
Input: A matrix $\mathbf{Z} \in \mathbb{R}^{n\times 2}$ of coordinates in the $2D$ instance space and a vector $\mathbf{y} \in \mathbb{B}^{n\times 1}$ of binary performance measures.
Output: A footprint $\Phi$.
Function TRACEbuild($\mathbf{Z}, \mathbf{y}$) is
    $\mathbf{Z}_u \leftarrow$ UniquePoints $(\{\mathbf{z}_i | y_i = \text{TRUE}\})$;          // Find the unique points $\mathbf{Z}_u$ that have $y_i = \text{TRUE}$
    $\{r, c\} \leftarrow$ MatrixSize $(\mathbf{Z}_u)$;          // Find the number of rows and columns of the matrix
    $k \leftarrow \max(\min(\lceil r/20 \rceil, 50), 3)$;
    $\varepsilon \leftarrow \left(k\Gamma(2)/\sqrt{r\pi}\right)(\text{range}(\mathbf{z}_1) \times \text{range}(\mathbf{z}_2))$;
    $\mathbf{c} \leftarrow$ DBSCAN $(\mathbf{Z}_u, k, \varepsilon)$;          // Use DBSCAN to identify outliers and clusters of dense data
    $\Phi$.polygon $\leftarrow \emptyset$
    for $i = 1$ to $\max(\mathbf{c})$ do
        // For every detected cluster, build an $\alpha$-shape
        $\Phi$.polygon $\leftarrow$ JoinPolygons $(\Phi.\text{polygon}, \text{BuildAlphaShape}(\{\mathbf{z}_i | c_i = i\}))$;
    end
    $\Phi$.area $\leftarrow$ FindPolygonArea $(\Phi.\text{polygon})$;
    $\Phi$.density $\leftarrow$ CountElements $(\Phi.\text{polygon}, \mathbf{Z})\,/\Phi.\text{area}$;
    $\Phi$.purity $\leftarrow$ CountElements $(\Phi.\text{polygon}, \{\mathbf{z}_i | y_i = \text{TRUE}\})\,/$CountElements $(\Phi.\text{polygon}, \mathbf{Z})$;
end
```

Algorithm 8: TRACEcomp: Footprint comparison algorithm

```
Input: A base and test footprints $\{\Phi_B, \Phi_T\}$, a matrix $\mathbf{Z} \in \mathbb{R}^{n\times 2}$ of coordinates in the $2D$ instance space and two vectors
       $\mathbf{y}_B, \mathbf{y}_T \in \mathbb{B}^{n\times 1}$ of binary performance measures.
Output: A base and test footprints $\{\Phi_B, \Phi_T\}$ with removed contradictions.
Function TRACEcomp($\Phi_B, \Phi_T, \mathbf{Z}, \mathbf{y}_B, \mathbf{y}_T$) is
    $C \leftarrow$ PolygonIntersection $(\Phi_B.\text{polygon}, \Phi_T.\text{polygon})$;
    $N_{\text{try}} \leftarrow 0$;
    $N_{\max} \leftarrow 3$;
    while FindPolygonArea $(C) > 0 \wedge N_{try} < N_{\max}$ do
        $\pi_B \leftarrow$ CountElements $\left(C, \{\mathbf{z}_i | y_{B,i} = \text{TRUE}\}\right)/$CountElements $(C, \mathbf{Z})$;
        $\pi_T \leftarrow$ CountElements $\left(C, \{\mathbf{z}_i | y_{T,i} = \text{TRUE}\}\right)/$CountElements $(C, \mathbf{Z})$;
        if $\pi_B > \pi_T$ then
            $\Phi_T$.polygon $\leftarrow$ RemovePolygon $(\Phi_T.\text{polygon}, C)$;
        else if $\pi_B < \pi_T$ then
            $\Phi_B$.polygon $\leftarrow$ RemovePolygon $(\Phi_B.\text{polygon}, C)$;
        else
            break;
        end
        $C \leftarrow$ PolygonIntersection $(\Phi_B.\text{polygon}, \Phi_T.\text{polygon})$;
        $N_{\text{try}} \leftarrow N_{\text{try}} + 1$;
    end
    $\Phi_B$.area $\leftarrow$ FindPolygonArea $(\Phi_B.\text{polygon})$;
    $\Phi_B$.density $\leftarrow$ CountElements $(\Phi_B.\text{polygon}, \mathbf{Z})\,/\Phi_B.\text{area}$;
    $\Phi_B$.purity $\leftarrow$ CountElements $(\Phi_B.\text{polygon}, \{\mathbf{z}_i | y_i = \text{TRUE}\})\,/$CountElements $(\Phi_B.\text{polygon}, \mathbf{Z})$;
    $\Phi_T$.area $\leftarrow$ FindPolygonArea $(\Phi_T.\text{polygon})$;
    $\Phi_T$.density $\leftarrow$ CountElements $(\Phi_T.\text{polygon}, \mathbf{Z})\,/\Phi_T.\text{area}$;
    $\Phi_T$.purity $\leftarrow$ CountElements $(\Phi_T.\text{polygon}, \{\mathbf{z}_i | y_i = \text{TRUE}\})\,/$CountElements $(\Phi_T.\text{polygon}, \mathbf{Z})$;
end
```

To construct an algorithm's footprint, Algorithm 7 uses DBSCAN [12] to identify high density clusters of good instances. As outputs, DBSCAN provides a vector $\mathbf{c} \in \{-1, 1, \ldots, N_c\}$ with one entry per good instance, where '-1' marks an outlier, and values in $[1, N_c]$ range correspond to the index of the identified clusters. DBSCAN requires two parameters, $\{k, \varepsilon\}$, with the former representing the minimal number of neighboring instances to be considered a cluster, and the latter corresponding to the neighborhood radius. DBSCAN has been shown to be robust to a variety of parameter values [35], therefore both of them are automatically chosen following the equations [9]:

$$k \leftarrow \max\left(\min\left(\lceil r/20 \rceil, 50\right), 3\right) \tag{8}$$

$$\varepsilon \leftarrow \frac{k\Gamma\left(2\right)}{\sqrt{r\pi}}\left(\mathrm{range}\left(\mathbf{z}_1\right) \times \mathrm{range}\left(\mathbf{z}_2\right)\right) \tag{9}$$

where r is the number of unique instances in the space with good performance, and $\Gamma\left(\cdot\right)$ is the Gamma function. The footprint is then constructed using an α-shape, a generalization of the concept of convex hull from computational geometry [11], which corresponds to a polygon that tightly encloses all the points within a cloud. An α-shape is constructed for each cluster, and all shapes are bounded together as a MATLAB polygon structure.

Once constructed, overlapping footprints can appear when two algorithms simultaneously claim to be the best in an area of the instance space. To focus on the most compelling evidence for best performance, these overlapping sections are removed from the footprint with lower purity using Algorithm 8. The process is repeated $N_{\max} = 3$ times, although more than a single pass is often unnecessary. If the purity is the same for both footprints, the overlapping section is kept, as there is insufficient evidence of dominance of either algorithm.

3.5 Analyzing the instance space

Once the instance space has been constructed, and the algorithm footprints generated using Algorithms 1 to 8, ISA can commence to drive the intended insights: adequacy of benchmark instances, suitability of chosen features, and objective assessment of the strengths and weaknesses of various algorithms under different conditions represented by instance features. Moreover, the data generated by the ISA Toolkit is stored in CSV files, allowing researchers to further explore the results through alternative means, such as box and whiskers diagrams.

3.5.1 Distribution of Instance Sources. The first figure that should be analyzed is the distribution of various sources of instances across the instance space, which is identified as `'distribution_sources'`. Typically, randomly generated instances will exhibit a lack of diversity and be located around the center of the instance space, with average feature values. Other sources of instances possessing structures that create particular challenges will most likely occupy unique regions of the instance space. It is often interesting to observe how the benchmark test instance have evolved historically by tracking their movement across the instance space, typically in the direction of hardness, as instances that were once challenging become easier to solve over time with advances in algorithms and computing power. Other observations worth noting from these outputs are the theoretical or experimental boundary of the instance space, and the extent to which the instances in the collected meta-data span the region of possible instances determined by this boundary. Holes in the instance space, where no evidence has yet been collected of algorithm performance, should also be noted and addressed in subsequent meta-data augmentation during the iterative ISA process. Finally, the location of any real-world instances is also revealed if they have been labeled in the Source column identifier (see Figure 2), and will be important for subsequent analysis of algorithm suitability for real-world deployment.

3.5.2 Automated Algorithm Selection. The degree to which the instance space is a satisfying representation of the instances, and is capable of meaningful insights into algorithm performance, depends on the extent to which the instance features have sufficiently captured the inherent difficulty of the instances and are predictive of algorithm performance. To this end, it is useful to next consider the accuracy of the machine learning models used for automated algorithm selection in the toolkit. This can be achieved by examining both the actual and predicted binary performance visualizations, which are identified with the string `'binary_performance_'` or `'binary_svm_'` followed by the algorithm name, and a CSV file containing the performance of the models identified as `'svm_-table'`. If the accuracy of each SVM model is reasonable, and visually we can confirm that the predicted good performance regions appear to match the evidence of good performance across the instance space, we can be confident that the selected features must be sufficient to represent the diversity of instances and their impact on algorithm performance. If not, the feature set is most likely insufficient, and new features should be considered to explain the performance of some algorithms. The accuracy of each algorithm's SVM model will indicate which algorithm's performance is not well explained by the selected features.

3.5.3 Algorithm Footprints. Next the algorithm footprints can be inspected to identify which algorithms have unique behaviors, and which regions of the instance space are easy and hard for different algorithms, and the portfolio as a whole. The directions of hardness are usually quite clear from these footprint graphs, since the easy region of the instance space will contain all algorithm footprints, while the instances at the opposite ends of the instance space are usually served by fewer algorithms (and sometimes none). In addition to insights offered by the visualization of algorithm footprints across the instance space, the footprint metrics calculated by the toolkit and stored in a CSV file identified as `'footprint_performance'` provide an objective assessment of the strength of evidence of good performance. The footprint density metrics can also inform analysis of which regions of the instance space require more instances to increase statistical support of conclusions, and where over-representation of certain types of instances may be contributing to bias that can potentially affect the machine learning models, and is hidden by traditional statistical analysis.

3.5.4 Distribution of Features. Beyond understand which algorithm is predicted to perform well in certain regions of the instance space, one of the most powerful insights afforded by the visualization of the instance space is to understand why. With the easy and hard regions of the instance space identified, and each algorithm's footprint generated, the distribution of each feature across the instance space can be inspected to explain how certain instance properties contribute to ease or difficulty for each algorithm.

3.5.5 Insights. Combining all of the above analysis provides an opportunity —if the meta-data is sufficient— to gain insights into the unique strengths and weaknesses of each algorithm, and the kind of instances that possess the properties that each algorithm can exploit for competitive advantage. Such insights will be illustrated in Section 4.

3.6 Generating additional meta-data

The initial instance space may not yield sufficient insights, due to the limitations of the meta-data. This may be either due to: (a) the lack of diversity of the test instances, or their bias towards suiting some algorithms more than others; (b) the algorithm footprints being quite similar to each other, either due to a very similar underlying mechanism guiding their behavior, or where they are quite different algorithms the similar footprints may be a limitation of the instances failing to identify any unique strengths and weaknesses; and (c) variations in algorithm performance may not be explainable in terms of the characteristics of instances if the feature set cannot adequately distinguish between good and not-good performance. In the latter case, the support vector machines are not expected to perform well if the available features are not highly predictive of algorithm performance.

An analysis of any limitations of the meta-data that have restricted the quality of insights possible through ISA will lead to new ideas about augmentations of the meta-data with additional instances, algorithms and features, enriching the potential to learn more. Some of this analysis is facilitated by the `'exploreIS'` routine.

3.6.1 New test instances. The analysis of the instance sources may identify gaps in the instance space that none of the current instance classes occupy. If these gaps correspond to regions where the SVM models are uncertain of algorithm performance, due to lack of evidence, it may be important to generate new instance to fill such gaps. Exploring additional parameter variations for generators that produced instances nearby may be an effective first step. Alternatively, a Genetic Algorithm can be used to set target points in the instance space and evolve existing instances to adapt their data so that they occupy new locations in the gaps [39]. Regardless of how new instances are created or sourced, the toolkit facilitates the projection of new instances into the instance space through the `'exploreIS'` routine, which takes a new CSV file, named `'metadata_test'`, with the same structure as shown in Figure 2 and each row corresponding to a new or an existing instance.

3.6.2 New algorithms. Observing the algorithm footprints for the portfolio of algorithms considered in the meta-data may reveal too many similar algorithms, or the need to add new algorithms that operate in a fundamentally different manner. The insights into how algorithms are affected by instance features may also create inspiration for new algorithm ideas or modifications. Additional algorithms can be added into the instance space by adding to the `'metadata_test'` file extra columns identified with the prefix `'algo_'`.

3.6.3 New features. Analysis of the SVM model performance may also reveal that some algorithm performances are not well explained by the selected features. Considering the differentiating mechanism employed by an algorithm, and what kind of instances would make this successful or not, may suggest new feature ideas. Additional features can be added into the meta-data file and an updated instance space generated. If the SVM models improve in accuracy, it is likely that the new features are valuable, and additional insights will be afforded in the second iteration.

Once the meta-data has been augmented with additional instances, algorithms or features, the iterative loop of ISA commences with a return to Step 2 to generate a more comprehensive new instance space for further analysis and insights. We now illustrate one iteration of ISA using a case study, whose meta-data is downloadable for reproducibility, and instance space available as a MATILDA library problem for further visual exploration[3].

4 CASE STUDY - INSIGHTS INTO UNIVERSITY TIMETABLING ALGORITHMS

The curriculum-based course timetabling problem was the subject of track 3 of the 2007 International Timetabling Competition (ITC2007). 21 real-world instances from the University of Udine were available. In a series of studies [22, 23, 41] we have taken two of the top performing algorithms in ITC2007 [4] - simulated annealing with constraint propagation (SACP) [31] and an unpublished algorithm based on tabu search over a weighted constraint satisfaction problem (TSCS) - and examined their performance on instances from a variety of sources. Besides the 21 competition real-world instances, we used instances from a random generator of Burke et al. [6], and instances we generated by using machine learning [23] to adapt the parameters of this random generator to create instances that are i) real-world like, and ii) more discriminating of unique algorithm performance [2]. A total of 8199 instances from these three classes are therefore available, along with two highly competitive algorithms, SACP and TSCS.

A standard performance analysis such as shown in Table 1 would conclude that both algorithms perform quite similarly, with SACP best on 26% of the instances, TSCS best on 29% of the instances, and equal performance on 45% of the instances. TSCS is better on the randomly generated instances, although most of them elicit tied performance, and SACP is slightly better on the real-world like instances [23]. Performance on the Udine real-world instances is mixed, and therefore inconclusive as to which algorithm is better for these particular real-world instances.

[3]See https://matilda.unimelb.edu.au/matilda/problems/opt/tt
[4]See http://www.cs.qub.ac.uk/itc2007

Table 1. Standard statistical analysis of timetabling case study performance results

Instances	SACP best	TSCS best	Tied	Total
Random	475	957	3060	4492
Real-world-like	1613	1442	631	3686
Udine	8	10	3	21
Total	2096	2409	3694	8199
%	25.56%	29.38%	45.05%	100%

We now show the kind of insights that can be gained through ISA.

4.1 Meta-data

With these 8199 instances and two algorithms, we now need to establish the performance metric to define good performance, and the set of features used to construct an instance space. Since the objective function being minimized by each algorithm is the total number of conflicts of the best solution, we therefore define this as the performance metric, and an algorithm's performance is labeled as "good" if it produces the smallest number of conflicts relative to the other algorithm. Alternatives to this approach, enabled by setting different parameters in the MATLAB code, would be to define good in absolute terms (e.g. below a threshold number of conflicts), or to define as good an algorithm's performance if is within ϵ% of the best algorithm's performance. In this case study, we set $\epsilon = 0$ to demonstrate how ISA can reveal the conditions under which each algorithm is superior to the other, and we therefore define good performance to be the best performance. This is not to say that a worse performance than best is bad, but merely that we define good to be best for this case study, and that definition could be relaxed in subsequent analysis.

For the features, we use the same hardness metrics that we have used in earlier studies of this problem [22, 23, 41], based on a review of the properties that make timetabling difficult [42]. In addition to basic features (like number of events to be timetabled), we use features related to landmarking (obtained by running the DSATUR algorithm [8], which is optimal for bipartite graphs); features related to student and teacher conflicts, thus related to the underlying Graph Coloring problems, and features that come from the timetabling context, such as the degree of slack (available seats in rooms less the required seats). The full set of 32 features is described in Table 2.

The meta-data we use for the following ISA is therefore summarized as follows:

- **I**: 8199 instances from the three classes defined as Udine from ITC2007 (21 instances), Real-world-like [23](3686 instances), and Randomly generated [6](4492 instances);
- $\mathcal{F}$: 32 features described in Table 2;
- $\mathcal{A}$: 2 competitive algorithms from ITC2007, SACP and TSCS;
- $\mathcal{Y}$: number of conflicts, with good performance defined as an algorithm that is relatively best (minimal conflicts) with $\epsilon = 0$.

4.2 Creating an instance space using MATILDA

After running the PRELIM algorithms, the feature selection process described by the SIFTED algorithm results in a set of 7 features being selected as those most correlated with algorithm performance. The PILOT algorithm takes these 7 features to derive the optimal linear transformation to the $2d$ instance space, given by Equation (10):

Table 2. Set of initial 32 features used in the meta-data. For features that are computed for every node of a graph, both the mean and standard deviation of the resulting distribution are used.

Feature name	Description
Size related features: those that define the dimension of the problem (3 features)	
Number of Courses	Number of courses, independent of the number of lectures (events) per course.
Number of Events	Sum of all lectures across all courses.
Number of Rooms	Total number of rooms available.
Landmarking features: obtained from the DSATUR algorithm [8] (2 features)	
DSATUR Solution	Upper bound on the number of colors (period required for no conflicts)
DSATUR Color Sum	Sum of color values over all nodes (DSATUR objective function)
Graph Coloring features: from each of the conflict graphs $G(V, E)$, where V is a course, and E is a conflict between two courses, generated by: the curricula; the teacher availability; and the combination of both constraints (21 features)	
Edge Density	$\frac{\lvert E\rvert}{(\lvert V\rvert-1)^2}$
Node Clustering Index [3] mean and standard deviation	For each node $v \in V$, the edge density of the graph induced by v and its immediate neighbors.
Unweighted Event Degree mean and standard deviation	The degree of each node v.
Weighted Event Degree mean and standard deviation	The sum of the enrollments of all neighbors of v.
Timetabling features: features that come from the constraints unique to timetabling, as opposed to generic Graph Coloring features (6 features)	
Slack	Total seats in all the rooms less total seats required by all courses.
One Room Events	Number of events that will only fit in one room
Event Size mean and standard deviation	Number of students in each course
Room Options mean and standard deviation	The number of rooms into which each course can fit without penalty.

$$\mathbf{Z} = \begin{bmatrix} -0.9785 & 0.3046 \\ 0.8996 & -0.2604 \\ 0.7188 & 1.1367 \\ -0.0952 & 0.5218 \\ -0.4607 & 0.0560 \\ -0.6749 & 0.3208 \\ 0.6715 & 1.0739 \end{bmatrix}^{\top} \begin{bmatrix} \text{colorSum} \\ \text{eventDegreeTeacherConnectivity} \\ \text{meanRoomOptions} \\ \text{nCourses} \\ \text{nEvents} \\ \text{nRooms} \\ \text{slack} \end{bmatrix} \tag{10}$$

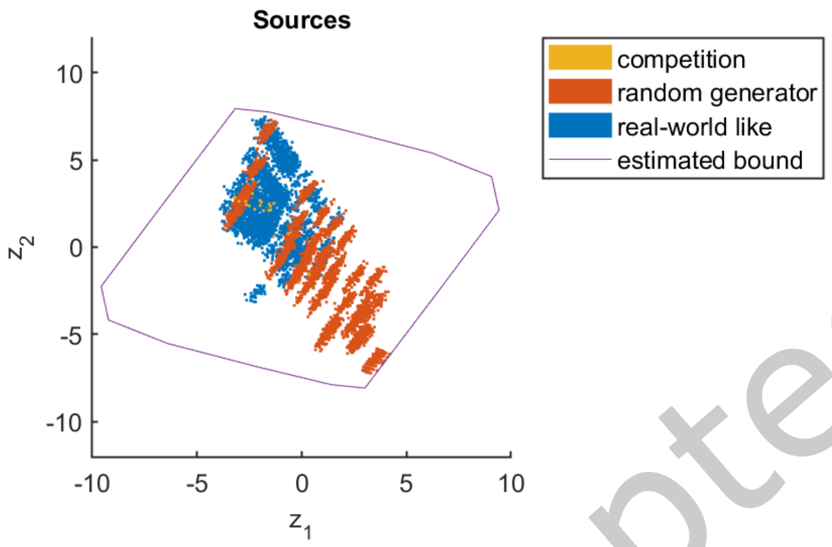


Fig. 3. Timetabling instance space defined by Equation (10) showing three sources of instances, and estimated boundary of instance space

Each timetabling test instance is thus projected to a location in the $2d$ instance space given by coordinate values $\mathbf{Z} = (\mathbf{z}_1, \mathbf{z}_2)$ by calculating its $7d$ feature vector and applying the linear transformation given by equation (10). The two axes of the instance space z_1 and z_2 are therefore linear combinations of the selected features. The SIFTED algorithm has identified features related to the underlying graph coloring problem (the sum of the colors required by the DSATUR heuristic to color the graph, and the mean node degree of the teacher connectivity conflict graph), as well as features related to the size of the problem (number of courses, events and rooms available), and the tightness of constraints (slack, and the mean number of room options for courses). Examining the coefficients of the linear transformation matrix in Equation (10), reveals that larger instances are pushed to the left (by the negative coefficients of z_1), and more tightly constrained (harder) instances lie along the bottom (by the negative coefficient of more teacher conflicts creating smaller values of z_2).

4.3 Distribution of Instances

The 8199 timetabling instances, projected to this instance space, are shown in Figure 3, colored by the source of the instances. It is clear that the 21 real-world Udine University instances used in the ITTC2007 competition (in yellow) have very different features compared to the randomly generated instances of Burke et al. [6] (in red). The instances that we have previously generated [23] by adapting the random generator to be more real-world like and discriminating (shown in blue), have more similarity to the real-world Udine instances while enabling the unique strengths and weaknesses of each algorithm to be more apparent, as discussed in the subsequent analysis of algorithm performance.

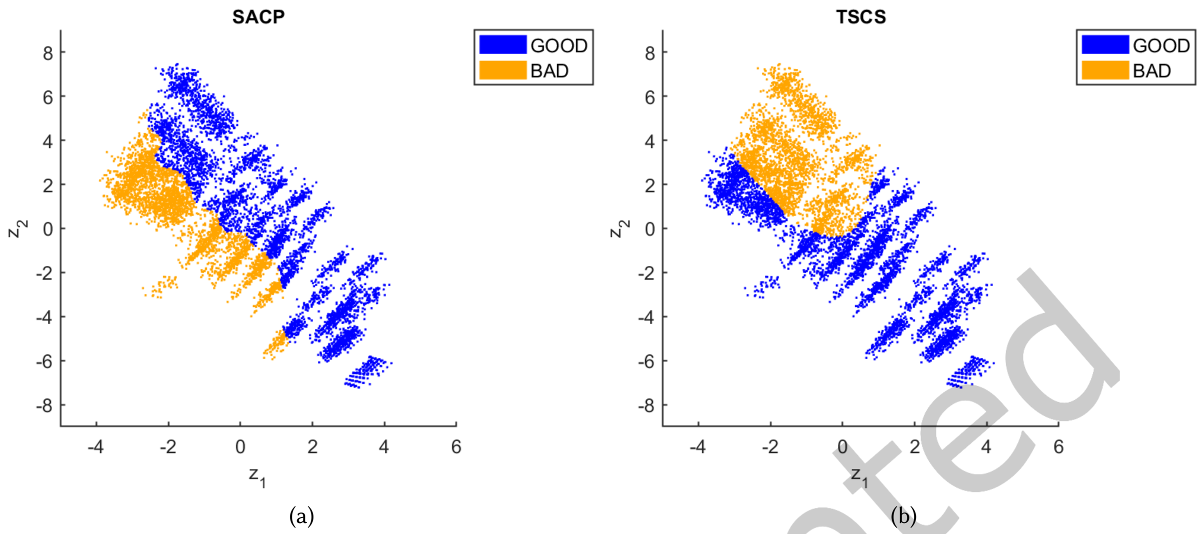


Fig. 4. SVM predicted performance of (a) SACP and (b) TSCS, where good performance means relatively best (minimal constraint violations) compared to the other algorithm.

4.4 Automated Algorithm Selection

A Support Vector Machine (SVM) is used to learn in which regions of the instance space we can expect “good” performance for each algorithm. Figure 4 shows the SVM predictions, with good performance predicted from both algorithms in the lower right region of the instance space, bad performance predicted around the location of many of the real-world Udine instances (insufficient evidence of consistently good performance for either algorithm in that region), and some unique strengths identified along the top edge (for SACP) and bottom edge (for TSCS). Combining these SVM models into a single algorithm selection recommendation, Figure 5 shows which algorithm is expected to perform best, with ties broken by selecting the SVM with higher precision. Note the interior region where neither of these methods is recommended, due to lack of statistical evidence, and the region at the bottom where both methods are expected to perform identically, but SACP is recommended due to the higher precision (stronger evidence) of its SVM model.

The performance of the SVM prediction models is shown in Table 3. The very high precision figures (92.5% and 91.3% respectively for SACP and TSCS) confirm that if an SVM declares that its algorithm will achieve good performance on an instance, it can be trusted. Recall is lower, indicating that some good performances may be missed by the SVMs, which are quite conservative. The overall accuracies are reasonable, but it is the precision that matters most for deciding if an SVM can be trusted to recommend an algorithm. The last two columns in Table 3 report the SVM parameters from the grid search procedure used in the PYTHIA algorithm. While the probability of correctly selecting the best algorithm for a given instance is only 70.6% if always using SACP, and 74.4% if always using TSCS, the SVM selector shown in Figure 5 has an 87.7% probability of success in predicting the best algorithm for a given instance. The performance metric (average number of student clashes $\overline{NC}$ across the set of all instances σ or the recommended subset σ_S) achieved by the selector is very close to the minimum achieved by an oracle with perfect information about which algorithm is best, and is improved compared to the performance metric achieved by either SACP or TSCS alone.

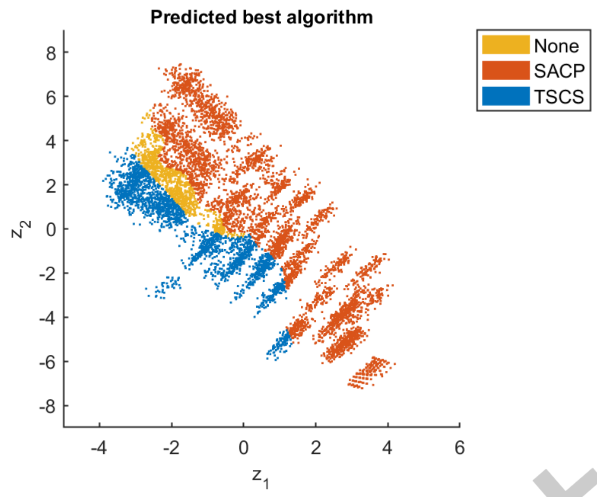


Fig. 5. Recommended algorithms from SVM prediction models, including region where neither algorithm has sufficient evidence for recommendation as best.

Table 3. SVM results

	$\overline{NC}$ (σ)	Pr (G)	$\overline{NC}_S$ (σ_S)	Accuracy	Precision	Recall	C	γ
SACP	959.183(1870.115)	0.706	546.863(1040.747)	81.0%	92.5%	79.6%	5.777	11.534
TSCS	949.931(1860.499)	0.744	1409.224(2186.322)	77.3%	91.3%	76.8%	0.023	5.11
Oracle	948.113(1861.310)	1.000						
Selector	948.940(1861.093)	0.877	1027.894(1942.289)		91.3%	56.1%		

4.5 Algorithm Footprints

Figures 6 and 7 show that both algorithms perform identically in the lower part of the instance space, where many of the randomly generated instances lie. These instances are not discriminating, either being equally hard or easy for both algorithms. TSCS is outperformed in the top of the instance space, largely where the real-world like instances lie, and the performance of both algorithms on the real-world Udine instances is mixed. It is clear that the randomly generated instances lying along the lower left edge of the region in the bottom right corner are equally hard for both algorithms, with solutions containing numerous constraint violations, while instances in the middle and upper edge of the bottom right region are easy for both algorithms. The improved discrimination of the real-world like instances is clear, compared to the tied performance on the randomly generated instances.

The footprints of the algorithms, calculated using the TRACE routine in Algorithm 6, are shown in Figures 8 and 9 for the actual observed performances. The areas, densities and purities of these footprints are shown in Table 4, considering both good, marked with subscript G, and best performance, marked with subscript B. On the former, ties are not broken, as more than one algorithm can be good simultaneously; whereas, in the latter, ties are broken at random, as there is no *a priori* reason why one algorithm should be labelled as outperforming another. All area and density values are normalized over the area of the space, understood as the convex hull enclosing the instances in **I**. Therefore, a footprint with density higher than 100% is denser than the space on average; while the opposite indicates that the footprint is sparser. We observe that for an area of close to one third of the space we either do not have enough instances, i.e., it has low density, or the performance of the

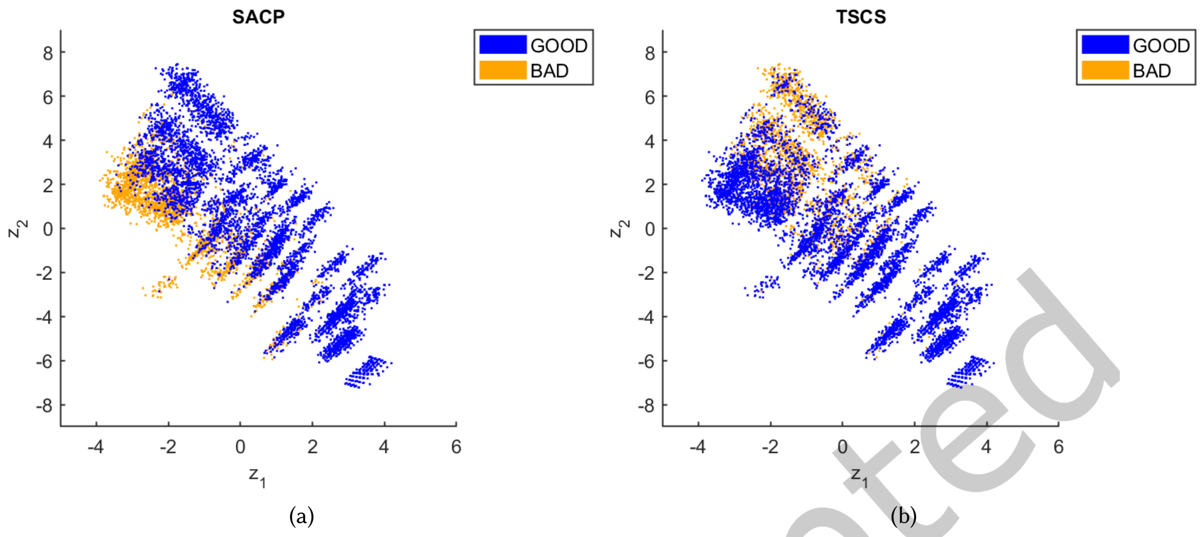


Fig. 6. Instances where each algorithm achieves the best performance in blue (minimal constraint violations) for (a) SACP and (b) TSCS, without breaking ties. Orange shows instances where the best performance was not obtained by each algorithm.

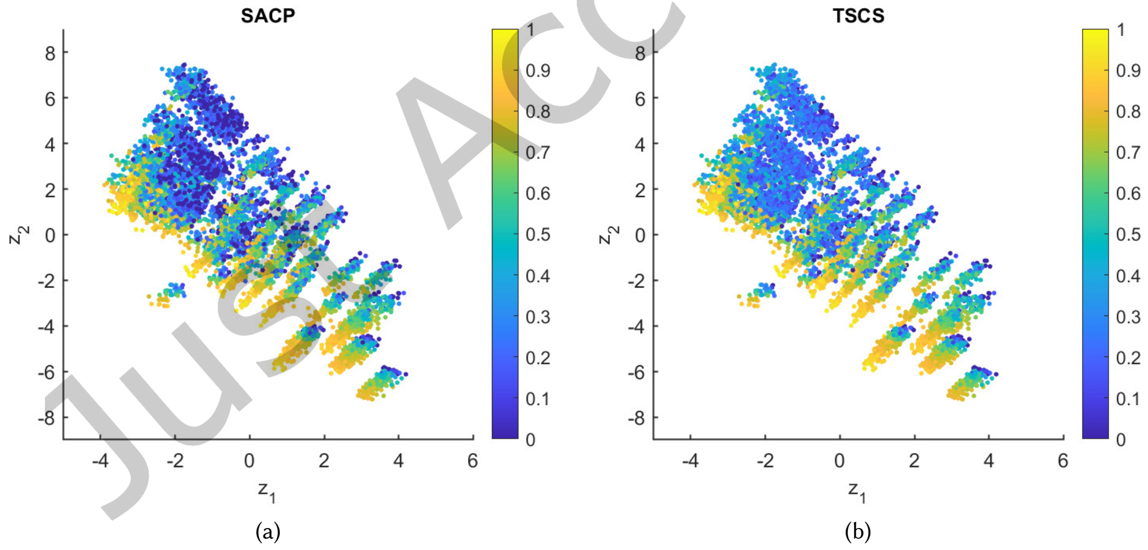
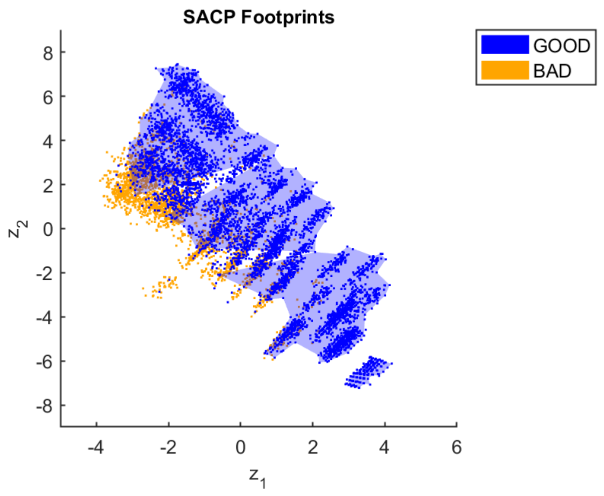
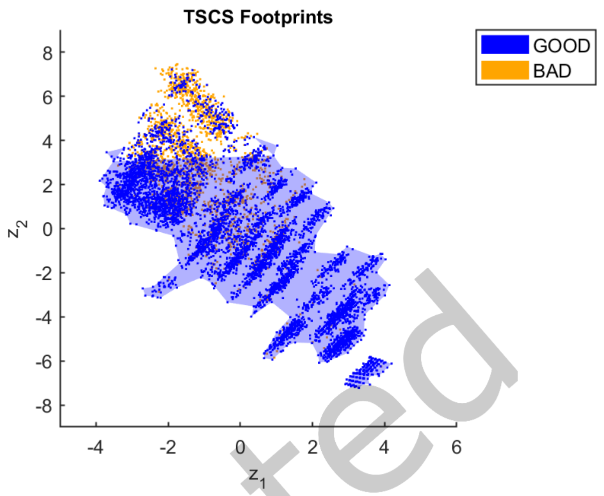


Fig. 7. Number of constraint violations achieved by each algorithm, (a) SACP and (b) TSCS. Data has been $\log_{10}$-transformed and scaled between the global maximum and minimum.

algorithms is so mixed that no best algorithm emerges, i.e., it has low purity. Although SACP has a larger unique



(a)



(b)

Fig. 8. Footprints for (a) SACP and (b) TSCS, where good performance means relatively best (minimal constraint violations) compared to the other algorithm. Ties were not broken.

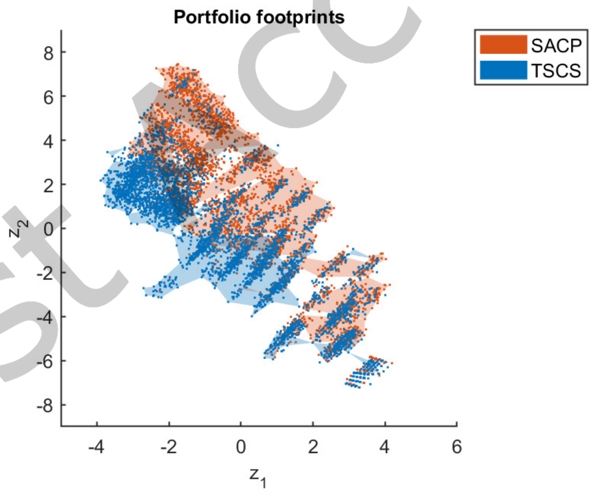


Fig. 9. Footprints where each portfolio is best (based on actual performance). Ties were broken randomly. Overlapping areas indicate that purity is the same for both algorithms, hence, not enough evidence exist that one of them dominates.

footprint, this is less dense, indicating that neither algorithm has a substantial unique advantage over the other. These results demonstrate the existence of complementary performance between these algorithms.

Table 4. Footprint metrics for SACP and TSCS timetabling algorithms

	$\alpha_{N,G}$	$d_{N,G}$	$p_{N,G}$	$\alpha_{N,B}$	$d_{N,B}$	$p_{N,B}$
SACP	77.3%	100.5%	81.5%	36.5%	89.1%	74.7%
TSCS	80.2%	103.7%	82.1%	29.6%	116.1%	73.0%

4.6 Distribution of Features

Now that we have established that each algorithm has unique areas of strengths and weaknesses, as well as similar performance for some instance classes, we seek to explain how the instance properties affect performance. Inspection of the distribution of features across the instance space in Figure 10 helps reveal the properties of instances in different regions and from different sources. We note that easier instances lie on the left side of the upper edge region, defined by higher slack value and more room options. These are from the real-world-like generator, and represent instances where SACP has an advantage. High values of the node degree of the teacher connectivity graph (eventDegreeTeacherConnectivity) defines the harder but non-discriminating instances in the bottom right region, which are also the smaller instances with fewer courses, events and rooms. Harder non-discriminating instances have less slack and fewer room options. Finally, we note that the real-world Udine instances have very low values of eventDegreeTeacherConnectivity (fewer teacher conflicts) compared to either of the synthetic generators, so this provides useful feedback to further refine the generator parameters to make it more real-world like, at least for the properties found in the Udine real-world context.

4.7 Insights into Algorithm Strengths and Weaknesses

Putting all of this analysis together, we reach the following conclusions:

- Udine instances have very different properties to the random generator: they are larger in scale, but with lower values of average node degree for the teacher connectivity graph, suggesting that teachers are involved in less classes at the University of Udine than the random generator assumes;
- Udine instances have moderate slack and meanRoomOptions values compared to the full range exhibited by the real-world-like generator [23];
- Both algorithms perform similarly on instances defined by higher values of node degree for the teacher connectivity graph, with lower values of node degree making instances more discriminating, and impacted by slack;
- SACP performs better when the node degree is low, and when slack is high;
- TSCS performs better for mid-range slack values, with both algorithms performing similarly for low slack values;
- SACP is good when slack is low (less than 200) and best when slack is high (above 400);
- TSCS is good when slack is less than 400, and best for slack in the range [200-400];
- There is too much inconsistency in which algorithm is best for instance in the vicinity of the real-world Udine instances. Consequently, the SVM models have less confidence in the prediction of which method is best in that central portion of the upper left region, although both algorithms could find low-cost solutions to these instances;
- Compared to the real-world-like instances, the Udine instances do not allow us to identify unique strengths and weaknesses of the two algorithms studied (they are easy for both, and if one doesn't beat the other, it is very close). Consequently, their value as discriminating benchmarks, at least for these two highly competitive algorithms, is limited.

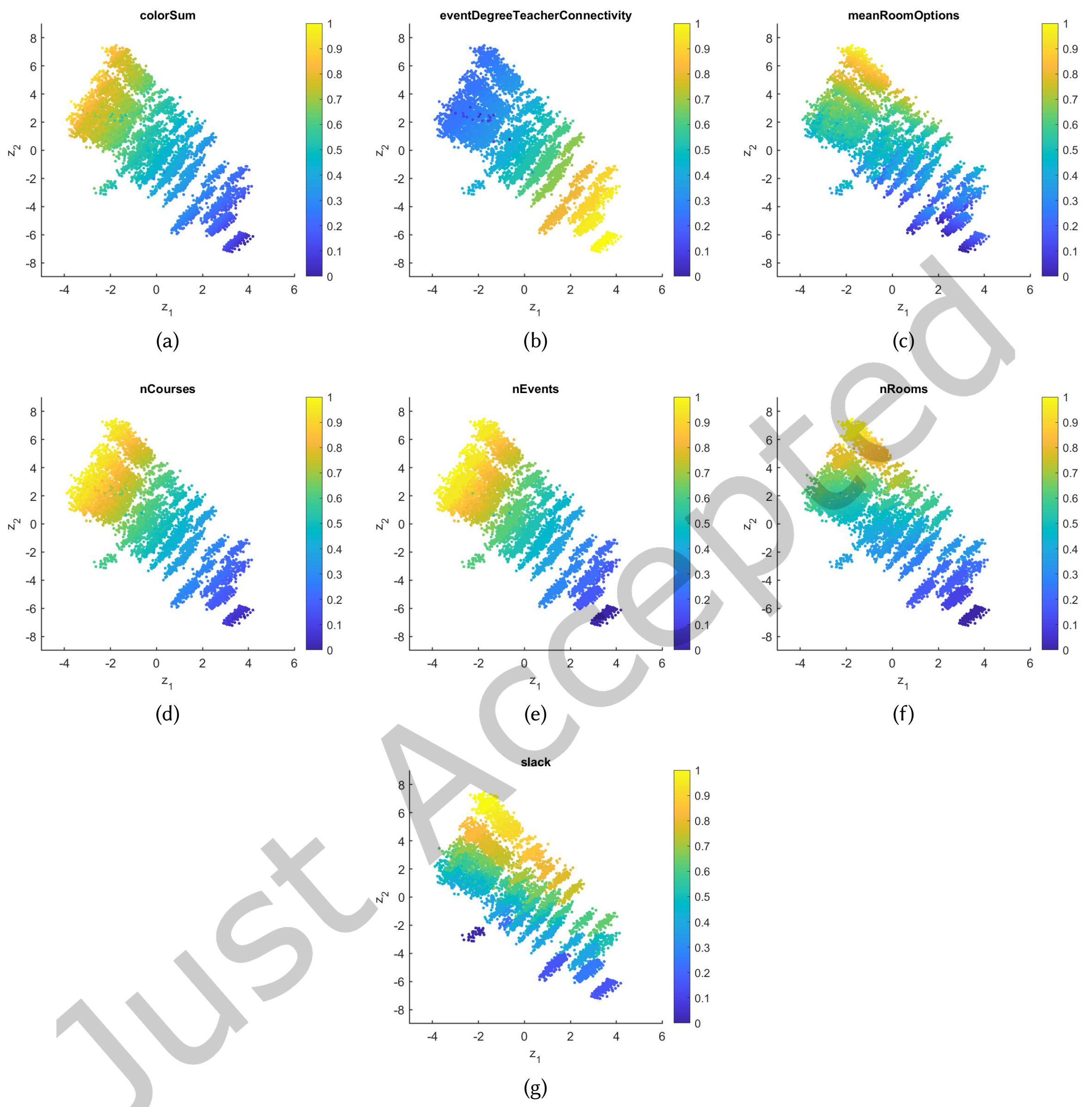


Fig. 10. Feature distributions for (a) colorSum, (b) eventDegreeTeacherConnectivity, (c) meanRoomOptions, (d) nCourses, (e) nEvents, (f) nRooms, (g) slack. Minimal values of each feature are indicated by coloring an instance blue, and maximal feature values shown as yellow.

It is interesting to note that this analysis broadly supports the findings in our previous study of this meta-data [41] (which used decision trees and self-organizing maps at that time). The decision tree analysis identified features such as slack that determine unique algorithm behavior, other features such as properties of the teacher connectivity graph that determine tied behaviors, and features that explain the key differences between the randomly generated instances and real-world instances. However, this ISA has allowed a more insightful understanding and visualization of how algorithm performance depends on the instances, compared to the machine learning approach used to generate rules in [41]. While the decision tree rules in [41] are supported by this analysis, they are not as comprehensive, with much subtle nuance being overlooked by machine learning methods if it only affects a small subset of instances. The conclusions from this ISA are also more nuanced compared to those generated by the initial statistical analysis of the performance data (see Table 1), which concluded that, on average, both methods are similar, with TSCS having a slight advantage.

4.8 Sufficiency of the Meta-Data

Following the initial ISA there is the opportunity to reflect on how augmentation of the meta-data could lead to further insights. The SVM prediction accuracies in Table 3 reveal that additional features may be required to capture the difficulty faced by both algorithms, particularly TSCS which had slightly worse accuracy, precision and recall. Considering the distinguishing mechanism employed by an algorithm, and how the properties of an instance may create challenges for this mechanism, can lead to new features being proposed. Adding additional real-world instances would also be recommended, particularly since the 21 Udine instances are quite different from the randomly generated and modified real-world like instances, and it would be worthwhile to establish if other real university instances fall in the same region or expand the instance space. While the two algorithms considered had some unique footprints, they are both very strong and shared similar footprints for much of the instance space (tied performance). Further to adding more real-world instances to expand the instance space, an augmented meta-data set would benefit from including additional algorithms, exploring a range of search strategies.

Indeed, such an augmented meta-data set has already been generated, based on an extension of this illustrative case study [10], with meta-data and ISA available as a library problem at https://matilda.unimelb.edu.au/matilda/problems/opt/tt In the extended work [10], the iterative nature of ISA is illustrated with the augmented meta-data constructed based on the insights afforded by the initial analysis presented in this case study. Specifically, a set of 1857 distinct features were considered to describe a comprehensive set of instances including 54 more real world instances, evaluated on state-of-the-art solvers (both exact and meta-heuristics), using a performance metric that considers solution quality as well as computational effort. Based on this comprehensive instance space, new instance generators were developed to fill the instance space and obtain new insights into the strengths and weaknesses of the chosen algorithm portfolio. We refer the interested reader to [10] and the corresponding ISA available as a MATILDA library problem for further details.

While the concept of “convergence" of the instance space is clear for a given set of algorithms - where there is no need for additional meta-data if the insights are sufficient and no new features are selected in the second iteration - in some sense the iterative nature of ISA continues as long as new algorithms are tested, or new instances are included that may require new features to be devised to capture their unique properties. For the purposes of this paper, however, we conclude with the initial ISA to illustrate the methods, and refer to a subsequent analysis with augmented meta-data [10] to illustrate the additional insights that become possible in subsequent iterations.

5 CONCLUSIONS

In this paper, we have presented the final methodology for Instance Space Analysis and its online software tools, developed over the last decade and now available for adoption by researchers at https://matilda.unimelb.edu.au.

We have argued for the need for improved methods for “stress-testing" algorithms in a more insightful way than traditional on-average statistical analysis affords. The conceptual framework of ISA has been presented as an extension of Rice’s Algorithm Selection Problem [34, 37], with the mathematical challenge of constructing a 2*d* instance space with its theoretical boundary discussed. The requirements of the meta-data have been described, including the formatting expectations for the software tools. The instance space is constructed using four methods (PRELIM, SIFTED, PILOT and CLOISTER) and the algorithms and their parameter settings have been presented. Once the instance space has been constructed, the analysis phase involves footprint analysis, machine learning predictions for automated algorithm selection, scrutiny of the instances for their diversity and bias, and assessment of whether additional meta-data is required for further insights to be obtained.

With the methodology and tools explained, the paper then presented an illustrative case study for the university curriculum-based course timetabling problem. The case study reveals how ISA offers a more nuanced understanding of the unique strengths and weaknesses of each algorithm in a portfolio, and is much more insightful than the initial conclusions about which algorithm is best from a standard statistical analysis. The opportunities revealed from this initial ISA have already been progressed with a further iteration considering a different portfolio of algorithms, with additional instances and features [10].

ISA is widely applicable to any field that conducts algorithm testing using test instances. The quality of benchmark test suites has been scrutinized using ISA in fields such as machine learning classification [30], black-box optimization [28, 29], time series forecasting [20], and various combinatorial optimization problems [40]. The list of available library problems at MATILDA's website [43] is growing, and currently contains many successful case studies to provide further illustration of how the ISA methodology can be applied to a wide variety of contexts: for testing algorithms, but also for testing model formulation and parameter configurations. We hope that this tutorial paper will complement MATILDA's growing collection of library problems, by providing the detailed explanation of the methods and tools, and support other researchers to adopt the ISA methodology to support rigorous "stress-testing" of algorithms and development of fit-for-purpose benchmark test suites.

While the ISA methodology has been successfully tested on many problems, it will undoubtedly continue to evolve and adapt in response to researcher suggestions of enhancements. Recent innovation have included a pre-processing stage to select a subset of instances to ensure the studied meta-data is free from representation bias, which can impact the selection of features, the projection algorithm, and the SVM automated algorithm selection results [1]. The challenge of proposing suitable features that correlate with algorithm difficulty has also seen a call for a feature-free implementation of algorithm selection approaches that could be extended to a feature-free instance space [2]. Finally, alternatives to the projection algorithm have been proposed with different criteria to determine the layout of the instance space [36]. New tools have also been developed to deliver the ISA methodology on different platforms, such as a Python tool with additional interface functionality to explore the instance space [33]. We welcome such innovations and look forward to continuing to expand the functionality and impact of ISA in the future.

ACKNOWLEDGMENTS

Funding was provided by the Australian Research Council under the Australian Laureate Fellowship scheme, grant number FL140100012, and through support of the ARC Training Centre in Optimisation Technologies, Integrated Methodologies and Applications (OPTIMA), grant number IC200100009. M.A. Muñoz was with the School of Mathematics and Statistics, The University of Melbourne, Parkville, VIC 3010, while conducting this work.

REFERENCES

[1] H. Alipour, M.A. Muñoz, and K. Smith-Miles. 2023. Enhanced instance space analysis for the maximum flow problem. *Eur. J. Oper. Res.* 304, 2 (2023), 411–428.

[2] M. Alissa, K. Sim, and E. Hart. 2019. Algorithm selection using deep learning without feature extraction. In *Proceedings of the Genetic and Evolutionary Computation Conference*. 198–206.

[3] C. Beyrouthy, E. K. Burke, D. Landa-Silva, B. McCollum, P. McMullan, and A. J. Parkes. 2009. Towards improving the utilization of university teaching space. *J. Oper. Res. Soc.* 60, 1 (2009), 130–143.

[4] P. Brazdil, C. Giraud-Carrier, C. Soares, and R. Vilalta. 2008. *Metalearning: Applications to data mining.* Springer.

[5] C. G. Broyden. 1970. The Convergence of a Class of Double-rank Minimization Algorithms 1. General Considerations. *IMA J. Appl. Math.* 6, 1 (1970), 76–90. https://doi.org/10.1093/imamat/6.1.76

[6] E. K. Burke, J. Mareček, A. J. Parkes, and H. Rudová. 2010. A supernodal formulation of vertex colouring with applications in course timetabling. *Ann. Oper. Res.* 179, 1 (2010), 105–130.

[7] C.C. Chang and C.J. Lin. 2011. LIBSVM: A library for support vector machines. *ACM Trans. Intell. Syst. Technol.* 2 (2011), 27:1–27:27. Issue 3.

[8] J. C. Culberson and F. Luo. 1996. Exploring the k-colorable landscape with iterated greedy. *Cliques, coloring, and satisfiability: second DIMACS implementation challenge* 26 (1996), 245–284.

[9] M Daszykowski, B Walczak, and D.L Massart. 2001. Looking for natural patterns in data: Part 1. Density-based approach. *Chemometrics and Intelligent Laboratory Systems* 56, 2 (2001), 83–92. https://doi.org/10.1016/S0169-7439(01)00111-3

[10] A. De Coster, N. Musliu, A. Schaerf, J. Schoisswohl, and K. Smith-Miles. 2022. Algorithm Selection & Instance Space Analysis for Curriculum-based Course Timetabling. *J. Sched.* 25 (2022), 35–58. https://doi.org/10.1007/s10951-021-00701-x

[11] H. Edelsbrunner, D. Kirkpatrick, and R. Seidel. 1983. On the shape of a set of points in the plane. *IEEE Trans. Inform. Theory* 29, 4 (1983), 551–559. https://doi.org/10.1109/TIT.1983.1056714

[12] M. Ester, H.P. Kriegel, J. Sander, and X. Xu. 1996. A density-based algorithm for discovering clusters in large spatial databases with noise. In *KDD '96*. AAAI Press, 226–231.

[13] M. Gallagher. 2016. Towards improved benchmarking of black-box optimization algorithms using clustering problems. *Soft Comput* 20, 10 (2016), 3835–3849. https://doi.org/10.1007/s00500-016-2094-1

[14] N. Hansen, A. Auger, R. Ros, O. Mersmann, T. Tušar, and D. Brockhoff. 2021. COCO: a platform for comparing continuous optimizers in a black-box setting. *Optim. Methods Softw.* 36, 1 (2021), 114–144. https://doi.org/10.1080/10556788.2020.1808977

[15] D.E. Hinkle, W. Wiersma, and S.G. Jurs. 2003. *Applied statistics for the behavioral sciences.* Houghton Mifflin.

[16] J.N. Hooker. 1994. Needed: An empirical science of algorithms. *Oper. Res.* 42, 2 (1994), 201–212.

[17] J. Hooker. 1995. Testing heuristics: We have it all wrong. *J. Heuristics* 1, 1 (September 1995), 33–42. https://doi.org/10.1007/BF02430364

[18] S. Kandanaarachchi, M.A. Muñoz, R. Hyndman, and K. Smith-Miles. 2019. On normalization and algorithm selection for unsupervised outlier detection. *Data Min. Knowl. Discov.* 34 (2019), 309—-354. https://doi.org/10.1007/s10618-019-00661-z

[19] S. Kandanaarachchi, M.A. Muñoz, and K. Smith-Miles. 2019. Instance space analysis for unsupervised outlier detection. (2019). 1st Workshop on Evaluation and Experimental Design in Data Mining and Machine Learning.

[20] Y. Kang, R.J. Hyndman, and K. Smith-Miles. 2017. Visualising forecasting algorithm performance using time series instance spaces. *Int. J. Forecast* 33, 2 (2017), 345–358. https://doi.org/10.1016/j.ijforecast.2016.09.004

[21] P. Kerschke and H. Trautmann. 2019. Comprehensive Feature-Based Landscape Analysis of Continuous and Constrained Optimization Problems Using the R-package flacco. In *Applications in Statistical Computing – From Music Data Analysis to Industrial Quality Improvement*, N. Bauer, K. Ickstadt, K. Lübke, G. Szepannek, H. Trautmann, and M. Vichi (Eds.). Springer, 93–123. https://doi.org/10.1007/978-3-030-25147-5_7

[22] L. Lopes and K. Smith-Miles. 2010. Pitfalls in Instance Generation for Udine Timetabling. In *Learning and Intelligent Optimization - 4th International Conference, LION 4, Selected Papers*, Christian Blum and Roberto Battiti (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 299–302.

[23] L. Lopes and K. Smith-Miles. 2013. Generating Applicable Synthetic Instances for Branch Problems. *Oper. Res.* 61 (06 2013), 563–577. https://doi.org/10.2307/23474003

[24] N. Macia and E. Bernadó-Mansilla. 2014. Towards UCI+: a mindful repository design. *Inform. Sciences* 261 (2014), 237–262.

[25] C.C. McGeoch. 2012. *A guide to experimental algorithmics.* Cambridge University Press.

[26] O. Mersmann. 2009. *Benchmarking evolutionary multiobjective optimization algorithms using R.* Master's thesis. Universitat Dortmund.

[27] M.A. Muñoz and K. Smith-Miles. 2017. Generating Custom Classification Datasets by Targeting the Instance Space. In *GECCO '17 (GECCO '17)*. ACM, New York, NY, USA, 1582–1588. https://doi.org/10.1145/3067695.3082532

[28] M.A. Muñoz and K. Smith-Miles. 2017. Performance analysis of continuous black-box optimization algorithms via footprints in instance space. *Evol. Comput.* 25, 4 (2017), 529–554. https://doi.org/10.1162/EVCO_a_00194

[29] M.A. Muñoz and K.A. Smith-Miles. 2021. Generating new space-filling test instances for continuous black-box optimization. *Evol. Comput.* 28, 3 (2021), 379–404. https://doi.org/10.1162/evco_a_00262

[30] M.A. Muñoz, L. Villanova, D. Baatar, and K. Smith-Miles. 2018. Instance Spaces for Machine Learning Classification. *Mach. Learn.* 107, 1 (2018), 109–147. https://doi.org/10.1007/s10994-017-5629-5

[31] T. Müller. 2009. ITC2007 solver description: a hybrid approach. *Ann. Oper. Res.* 172, 1 (2009), 429.

[32] M.A. Muñoz and K. Smith-Miles. 2020. *Instance Space Analysis: A toolkit for the assessment of algorithmic power.* https://doi.org/10.5281/zenodo.4750845

[33] P.Y.A. Paiva, C. Castro Moreno, K. Smith-Miles, M.G. Valeriano, and A.C. Lorena. 2022. Relating instance hardness to classification performance in a dataset: a visual approach. *Mach. Learn.* 111, 8 (2022), 3085–3123.

[34] J.R. Rice. 1976. The Algorithm Selection Problem. In *Advances in Computers*. Vol. 15. Elsevier, 65–118. https://doi.org/10.1016/S0065-2458(08)60520-3

[35] E. Schubert, J. Sander, M. Ester, H.P. Kriegel, and X. Xu. 2017. DBSCAN Revisited, Revisited: Why and How You Should (Still) Use DBSCAN. *ACM Trans. Database Syst.* 42, 3, Article 19 (July 2017), 21 pages. https://doi.org/10.1145/3068335

[36] K. Sim and E. Hart. 2022. Evolutionary Approaches to Improving the Layouts of Instance-Spaces. In *International Conference on Parallel Problem Solving from Nature*. Springer, 207–219.

[37] K.A. Smith-Miles. 2009. Cross-disciplinary perspectives on meta-learning for algorithm selection. *ACM Comput. Surv.* 41, 1 (2009), 6:1–6:25. https://doi.org/10.1145/1456650.1456656

[38] K. Smith-Miles, D. Baatar, B. Wreford, and R. Lewis. 2014. Towards objective measures of algorithm performance across instance space. *Comput. Oper. Res.* 45 (2014), 12–24. https://doi.org/10.1016/j.cor.2013.11.015

[39] K. Smith-Miles and S. Bowly. 2015. Generating New Test Instances by Evolving in Instance Space. *Comput. Oper. Res.* 63 (2015), 102–113. https://doi.org/10.1016/j.cor.2015.04.022

[40] K. Smith-Miles, J. Christiansen, and M.A. Muñoz. 2021. Revisiting *Where are the hard knapsack problems?* via Instance Space Analysis. *Comput. Oper. Res.* 128 (2021), 105184. https://doi.org/10.1016/j.cor.2020.105184

[41] K. Smith-Miles and L. Lopes. 2011. Generalising Algorithm Performance in Instance Space: A Timetabling Case Study. In *Learning and Intelligent Optimization - 5th International Conference, LION 5, Rome, Italy, January 17-21, 2011. Selected Papers*. Springer Berlin Heidelberg, 524–538. https://doi.org/10.1007/978-3-642-25566-3_41

[42] K. Smith-Miles and L. Lopes. 2012. Measuring instance difficulty for combinatorial optimization problems. *Comput. Oper. Res.* 39, 5 (2012), 875–889. https://doi.org/10.1016/j.cor.2011.07.006

[43] K. Smith-Miles, M.A. Muñoz, and Neelofar. 2020. *Melbourne Algorithm Test Instance Library with Data Analytics (MATILDA)*.

[44] K. Smith-Miles and T.T. Tan. 2012. Measuring Algorithm Footprints in Instance Space. In *Proceedings of the 2012 IEEE Congress on Computational Intelligence (CEC)*. 3446–3453.

[45] J. Vanschoren, J.N. van Rijn, B. Bischl, and L. Torgo. 2013. OpenML: Networked Science in Machine Learning. *SIGKDD Explorations* 15, 2 (2013), 49–60. https://doi.org/10.1145/2641190.2641198

[46] X. Wang, K. Smith, and R. Hyndman. 2006. Characteristic-based clustering for time series data. *Data Min. Knowl. Discov.* 13, 3 (2006), 335–364.

[47] D.H. Wolpert and W.G. Macready. 1997. No free lunch theorems for optimization. *IEEE Trans. Evol. Comput.* 1, 1 (Apr. 1997), 67–82. https://doi.org/10.1109/4235.585893

[48] S. Yarrow, K.A. Razak, A.R. Seitz, and P. Serès. 2014. Detecting and Quantifying Topography in Neural Maps. *PLoS ONE* 9, 2 (02 2014), 1–14. https://doi.org/10.1371/journal.pone.0087178