Machine scheduling instance generation by reverse engineering from instance space analysis

Christian Gahm , Michael Wimmer , Axel Tuma

PII: S0377-2217(25)00849-5
DOI: https://doi.org/10.1016/j.ejor.2025.10.029
Reference: EOR 19830

To appear in: *European Journal of Operational Research*

Received date: 4 February 2025
Accepted date: 17 October 2025

Please cite this article as: Christian Gahm , Michael Wimmer , Axel Tuma , Machine scheduling instance generation by reverse engineering from instance space analysis, *European Journal of Operational Research* (2025), doi: https://doi.org/10.1016/j.ejor.2025.10.029

Highlights

- The quality of algorithm performance testing strongly depends on the instances
- Instances need to be feasible, reasonable, diverse, and challenging
- Our instance generator is based on reverse engineering from instance space analysis
- Its benefits are demonstrated by three well-known machine scheduling problems

Machine scheduling instance generation by reverse engineering from instance space analysis

Christian Gahm[a*], Michael Wimmer[a], Axel Tuma[a]

[a] Chair of Business Administration, Production & Supply Chain Management, Augsburg University, D-86135 Augsburg, Germany

* Corresponding author:
E-mail: christian.gahm@uni-a.de; Phone: +49-821-598-4041

Abstract:

The availability of a sufficiently large number of meaningful instances for a scheduling problem is of utmost importance for the evaluation of solution methods for the problem. This study introduces a novel method for machine scheduling instance generation, termed Reverse Instance Generation (RIG), leveraging Instance Space Analysis. This method aims to create diverse, feasible, and realistic instances by reverse engineering from the instance space. Unlike existing approaches that rely on iterative search methods, RIG utilizes a constructive approach, combining dimensionality reduction techniques and controlled instance generation. The approach addresses the challenges of instance diversity and reasonableness, ensuring unbiased and reproducible outcomes. The effectiveness of RIG is demonstrated on three different machine scheduling problems: the single-machine weighted tardiness problem, the job shop scheduling problem, and a complex serial batch scheduling problem. The results highlight the method's ability to cover gaps in the instance space while maintaining practicality and efficiency, paving the way for improved benchmarking and algorithm development.

Keywords: instance generation, machine scheduling, instance space analysis
Declarations of interest: none

1 Introduction

For many years, researchers from various fields like Operations Research (OR) and its subfield machine scheduling have been developing solution methods to solve different kind of optimization problems. Hereby, the performance of solution methods is either assessed by deductive mathematical methods or by empirical analyses. As pointed out by Hooker (1994) in his seminal work "Needed: an empirical science of algorithms", deductive mathematical analyses provide only information about average-case or worst-case performances, which are often constrained by unrealistic assumptions about probability distributions or do not adequately reflect practical problems. Instead, he recommended to use a more empirical approach based on a rigorous experimental design with a diverse set of (problem) instances to test hypothesis about the performance of solution methods. This approach has become a standard for evaluating the performance of solution methods. However, in a later study, Hooker (1995) addressed two major problems associated with empirical performance analyses: In addition to implementation aspects of solution methods, the selection and/or generation of synthetic test instances presents several pitfalls to achieve an objective evaluation of solution methods. Since these statements, many researchers have studied the topic of selecting or generating appropriate instances, for example in the OR community, Kolisch et al. (1999), Hall & Posner (2001), Reilly (2009), and Smith-Miles & Bowly (2015). Based on this research, two questions need to be answered. First, what are appropriate instances and what properties must they have?

The appropriateness of instances is directly related to the purpose of the performance testing. According to Hall & Posner (2001), purposes can be

(P1) demonstrating the potential of a new solution method in specific situations,

(P2) demonstrating that a solution method is practical,

(P3) identifying conditions under which a solution method performs well or poorly, and

(P4) comparing competing solution methods.

The most important fundamental requirements derived from these purposes are that test instances have the right properties to (i) challenge algorithms and elaborate their strengths and weaknesses, and to (ii) reflect the characteristics of real-world problem instances (Smith-Miles & Bowly, 2015). A direct consequence of these requirements is that a set of test instances should be highly diverse. However, for many OR problems, synthetic and real-world instances exist but they are often not very different. This can lead to the temptation to design and tune solution methods to perform well on these instances, neglecting the performance on instances with diverse properties (Smith-Miles et al., 2014). In addition, many OR problems lack instance libraries or often contain only a small set of instances. This leads us to the second question: How to obtain an appropriate and comprehensive set of instances with the desired properties?

Generally, instances originate from real-world cases or are generated synthetically. As real-world instances are often not sufficient or comprehensive, instance generators provide the opportunity to overcome that problem. There are three basic types of instance generators known in the literature: The first type, naive instance generators, accomplish instance dissimilarity by drawing attributes from specific distributions. However, naive instance generators cannot guarantee to compute feasible instances (i.e., that all constraints of the optimization problem can be fulfilled by at least one solution; cf., Bowly et al., 2020). In addition,

they cannot guarantee for reasonable instances, i.e., that (functional) relationships between different elements of the data or value ranges are respected (e.g., the number of jobs to schedule should be greater than the number of machines, otherwise the problem becomes trivial; cf., Hall & Posner, 2010). To solve these problems, the second type generates instances by using "control attributes". Such "controllable instance generators" (CIGs) compute instances according to user-defined generation attributes, which typically define parameters of intervals and distributions, which in turn are used for randomizations. Through this interplay of attributes, parameters, and randomized values in the generation process, CIGs are able to generate reasonable and feasible instances with diverse properties. However, the diversity of such synthetic instances should be further improved (Bowly et al., 2020). To address the lack of diversity, advanced instance generators (AIGs), which either modify existing instance data or create entirely new instances, have been proposed in the literature. Most AIGs are based on the instance space analysis (ISA) framework, introduced by Smith-Miles et al. (2014), which visualizes the diversity of instances by projecting them onto a two-dimensional instance space. In general, AIGs use iterative meta-generation methods (e.g., Local search or Genetic algorithms) to search for new instances with some target properties or instances that are closest to target points in the instance space.

In contrast to such AIGs that search for new instances, we propose a more constructive approach: generating new instances through reverse engineering from ISA. This involves using the ISA method to project the existing instances belonging to the base sample and their high-dimensional feature vectors into a two-dimensional instance space via dimension reduction. Then, we calculate new desired features via inverse projection from artificially placed target points in the instance space. We use these reverse-engineered features to derive appropriate parameters for a CIG, which computes the new instances. By doing this, we establish a direct relationship between the target points, which represent specific instance characteristics, and the raw data of the new instance. This allows our reverse engineering instance generator (RIG) to produce a really diverse set of instances that is both feasible and reasonable due to the CIG application.

Our contributions to the literature are as follows:

- A new method (RIG) for generating diverse, feasible, and reasonable instances based on reverse engineering from instance space is proposed.
- The applicability of the new method is demonstrated by three problems from the literature: two well-known machine scheduling problems (the single machine weighted tardiness problem - SMTWTP and the classic job shop problem - JSSP) and a complex serial-batch scheduling problem with parallel machines, incompatible job families, sequence-dependent setup times, and arbitrary batch capacity requirements (PSBIJF).

Note that our proposed RIG method, in contrast to many approaches in the literature, is completely free of any bias towards preferring any solution method, as it intentionally ignores the performance of any solution method at all. In other words, we completely ignore the aspect of "hard" or "easy" solvable instances, because it is hardly possible to evaluate whether an instance is "hard" or "easy" to solve in an objective way based on the performance of existing solution methods. In our opinion, such a hardness assessment is always

based on a "snapshot" of the current state of research. A new solution method may easily solve instances that have been deemed "hard" thus far. The Combo algorithm (Martello et al., 1999) is a good example of such a “game-changing” solution method for the 1-0 knapsack problem (cf., Jooken et al., 2022). Another problem of focusing on hard instances and the often accompanying concentration on purpose P4 is that “the emphasis on competition is fundamentally anti-intellectual and does not build the sort of insight that in the long run is conducive to more effective algorithms” (Hooker, 1995). For these reasons, we take a very critical view of focusing on the generation of “hard” instances. However, if "hard" instances are available, these should definitely be included in the base sample to influence the generation of the new instances. Additionally, if only "hard" characteristics are known, a CIG with the corresponding attributes can be used to generate "hard" instances, which are then added to the base sample. Another option is to adjust the parameters of the CIG that computes the reverse-engineered instances.

The paper is structured as follows. First, the theoretical foundations of instance generation and instance space analysis are discussed in Section 2. In Section 3, the RIG method is presented, and in Section 4, the method is evaluated on the three problems from the literature. Our conclusions are presented in Section 5, together with suggestions for the use of RIG in the future.

2 Theoretical foundation and related literature

2.1 Theory of instance generation

According to Hall & Posner (2010), three major sources of data for the experimental analyses of solution methods exist: real-world data, library data, and random generation procedures (instance generators). Real-world data holds the advantage of its practical relevance, the resulting credibility for the experiment, and it is usually unbiased (e.g., does not favor any existing or new solution method). However, there is the drawback that real world data are not available in sufficient quantities for exhaustive experimental analyses. Moreover, most real-world data originate from the same (physical) process (e.g., demand data originate from the same supply policy) and there are situations where collecting real-world data is costly, lengthy, or inconvenient. If no suitable real-world data sets are available, library data could be used. For example, the OR-Library (Beasley, 1990) contains many instance sets for various OR and machine scheduling problems. An advantage of library data is that optimal or at least best-known solutions (objective values) are often available and comparisons between solution methods are transparent and verifiable. Nevertheless, library data does not exist for many real-world application cases. Furthermore, library data sets may be biased in such a way that a new solution method developed and published with the data set performs very well as the instances are not challenging for this solution method or any solution method at all (Hooker, 1995). For example, minimizing tardiness is not difficult when the due dates of a scheduling problem are very loose. To solve the problem of non-existent test instances for varying or new optimization problems, randomized instance generation procedures are often used to compute synthetic (artificial, academic) test instances. Sometimes, such procedures use existing real-world or library data and randomize them to obtain new instances (Smith-Miles & Bowly, 2015). However, not all the problems discussed can be solved in this way. Consequently, it may be better to generate entirely new synthetic instances that are not directly related to existing data but consider the “meta”-characteristics of real-world and library data (if appropriate). To

integrate these characteristics and further requirements (e.g., to challenge algorithms and explore their strengths and weaknesses), instance generators using control attributes have been developed. These control attributes are used by CIGs to define ranges for data values or to define distribution parameters that are used by a specified procedure to generate feasible and reasonable synthetic instances. However, all of the problems discussed so far can also arise in the context of CIGs and Hall & Posner (2010) outlined three principles (correctness, applicability, and reproducibility) and several generation properties for reducing and eliminating them:

Correctness: The data sets generated are free from defects.

Consistency: Identical types of data are generated in the same way.

The generator should not treat certain elements (e.g., the last one) of the instance set differently.

Unbiasedness: All biases in the data are controlled.

Instances should be generated in such a way that the data does not favor some experimental results (solution methods) over others. In terms of unbiasedness, it is helpful to think of all feasible instances as members of a population and generating a particular set of instances is analogous to selecting a sample from that population.

Applicability: The generator provides the types of data sets that are needed.

Completeness: All data sets that are important to the experiment can be generated.

Parsimony: The variations in the data sets are important to the experiment.

Variance in the data should be caused only by variations related to the experimental design (e.g., defined by control attributes) and the desired randomness.

Comparability: The experiments are comparable within and between studies.

Reproducibility: The generation process and its data sets are reproducible.

Describability: The generator is easy to describe.

Efficiency: The generator is easy and efficient to implement, use, and replicate.

Since recent developments in online data repositories (providing instance data and results) and the ability to make source code available online have made comparability and reproducibility much easier to achieve than in the past. Therefore, the most important and challenging properties to be achieved by instance generators are consistency, unbiasedness, completeness, and parsimony. Strongly related to these properties are the already discussed requirements of feasibility, reasonability, and diversity of instances. The last requirement for diversity in particular causes problems for CIGs if, for example, data intervals are too narrow, or distribution parameters are not set properly. To improve instance diversity, a new class of instance generators, so called advanced instance generators (AIGs) has been developed. However, the mere evaluation of diversity, which is a prerequisite for its improvement, is challenging. To this end, the instance space analysis (ISA) framework, first described by Smith-Miles et al. (2014), provides a very good visual and traceable method to assess the diversity of a set of instances. Since most existing AIGs are based on

ISA, as is our proposed method, we briefly describe it in the following.

2.2 Instance space analysis (ISA)

The ISA framework proposed by Smith-Miles et al. (2014) was originally developed to provide visual insights into the diversity and discrimination capabilities of instance sets. In many publications, the ISA is integrated into frameworks that address the "algorithm selection problem" (ASP; cf., Rice, 1976) and is used to visualize the performance of a solution method through "footprints" in the instance space (see, for example, Smith-Miles & Bowly, 2015 or Muñoz & Smith-Miles, 2017). As the ASP is not directly relevant for the development of AIGs, we omit its detailed description here but focus on the parts relevant for this study.

The basic idea of the ISA framework is the projection of an instance x from the "problem space" P (representing a possibly infinite set of instances of a problem; the set of existing real-world and/or synthetic instances is called $E \subseteq P$) and its properties expressed by numerical features from the "feature space" F (instance $x \in E$ is described by a feature vector calculated by function f) onto the two-dimensional "instance space" I^2 by a mapping p. Thus, we have a transformation of instances to points in $\mathbb{R}^2$: $E \xrightarrow{f} F \xrightarrow{p} I^2$ (note that the mapping p is nothing else but a dimension reduction and known methods can be used for it, e.g., Principal Component Analysis - PCA). The resulting points can then be visualized, and the diversity of instances easily be evaluated.

There are two important aspects to consider when using this framework: First, since P is generally unknown, a subset E must be used and its composition can have a strong influence on the shape of I^2. For example, if some authors have published very special instances, the question arises whether these should be considered or not (to avoid bias). Second, determining numerical features to adequately represent instances (i.e., feature engineering) is challenging, as it is in machine learning in general. In the context of ISA, the feature space F should contain not only simple properties, but also sophisticated ones with a greater ability to discriminate between instances (Smith-Miles et al., 2014). Hereby, multiple features measuring the same properties or with simple correlations should not be used simultaneously (i.e., in a single feature vector). Finding the most appropriate subset of features is most often an experimental process (for more information on feature selection in the context of ISA, see, for example, Muñoz et al., 2018). A comprehensive description of the ISA framework and related software tools can be found in Smith-Miles & Muñoz (2023).

2.3 Advanced instance generators

As discussed in the introduction, AIGs are designed to increase the diversity of instance sets while satisfying the requirements of consistency, unbiasedness, completeness, parsimony, feasibility, and reasonability. To analyze existing AIGs from the literature, we particularly focus on four aspects to identify the most relevant literature in the context of this paper: the purpose of instance generation (e.g., to generate hard to solve instances), the type of data generation (e.g., to modify existing instance raw data), the generation meta-method (e.g., a Genetic algorithm that searches for the new raw data), and the considered optimization problem. In Table 1, we give an overview of the analyzed literature. explicitly addressing the topic of advanced instance generation.

Table 1: Overview of related literature

Reference	Purpose				Data generation			Meta-method	Problem
	Max. diversity	Hard instances	Target properties	Target points in I^2	Modifying raw data	New raw data	CIG with attribute adaptation		
Cotta & Moscato (2003)		x			x			GA	Sorting
van Hemert (2006)		x			x			GA	CSP
Smith-Miles et al. (2010)		x			x			GA	TSP
Smith-Miles & van Hemert (2011)		x			x			GA	TSP
Lopes & Smith-Miles (2013)		x					x	LS	Timetabling
Mersmann et al. (2013)		x			x			GA	TSP
Smith-Miles & Bowly (2015) *				x	x			GA	Graph coloring
Kang et al. (2017) *				x	x	x		GA	Forecasting
Muñoz et al. (2018) *			x			x		GMM	Classification
Lou & Yuen (2019)		x[1]					x	GA	CBBO
Bowly et al. (2020) *			x		x		x	LS	Linear programs
Muñoz & Smith-Miles (2020) *			x	x		x		GP	CBBO
Gao et al. (2021)	x	x			x			GA	TSP
Smith-Miles et al. (2021) *				x	x		x	GA	Knapsack
Vela et al. (2021) *		x[1]	x			x		PSO	JSP
Coster et al. (2022) *			x				x	SbD	Timetabling
Shand et al. (2022)	x	x				x		GA	Clustering
Yap et al. (2022)	x				x		x	Man.[2]	continuous MOOP
Lechien et al. (2023)		x[1]				x		GA	HCP
Liu et al. (2023) *				x		x	x	GA	Bin packing
van Bulck & Goossens (2023) *				x			x	MILP	Sports timetabling
This study				x			x	RIG	Scheduling

Abbreviations: CBBO := Continuous black-box optimization, CSP := Constraint satisfaction problem, GA := Genetic algorithm, GMM := Gaussian mixture model, GP := Genetic programming, HCP := Hamiltonian completion problem, JSP := Job shop problem, LS := Local search, MILP := Mixed-integer liner program, MOOP := Multi-objective optimization problem, PSO := Particle swarm optimization, SbD := Sampling from estimated distributions, TSP := Traveling sales person

x[1]: Here, not hard but instances that are easy for one algorithm are desired.

Man.[2]: The new instances (i.e., the considered objective functions) are more or less "manually" created.

In the following, we give a more detailed analysis of the papers with the purpose of "Target properties" or "Target points in I^2" as these are the most relevant for our approach (marked with * in Table 1).

As first, Smith-Miles & Bowly (2015) developed an AIG that used a GA (Genetic Algorithm) to evolve randomly generated instances and adapt their raw data until they are located at target points in the

instance space. The starting point of the GA search was the "valid instance space", which resulted from the linear projection (via PCA) of a set of inequalities (constraining the high-dimensional feature space) onto the two-dimensional instance space. Their GA used four strategies to fill the valid instance space: First, the GA fitness function rewards instances with increasing distance from the known instances. Second, a grid of target points in the valid instance space is defined and the fitness decreases with increasing distance from the target point. Third, target points are defined along the theoretical boundary of the valid instance space. Fourth, target points that are arbitrarily close to known instances but well distributed throughout the valid instance space are determined, and not only instances but also target points have been evolved by the GA. Using these four strategies, a new set of 8,278 instances was generated for the graph coloring problem. The authors reported several findings. First, the evolved instances came much closer to the boundaries of the valid instance space than the existing ones. Second, with the first strategy, the GA became trapped in a limited set of regions and that strategies two and three resulted in many instances that were very far from any known instances. Third, strategy four achieved a good compromise between the exploration and the exploitation of the valid instance space.

Kang et al. (2017) used a GA for evolving time series instances. They applied a 32×32 rectangular grid with 1,024 target points determined by minimum and maximum values one unit wider than the upper and lower bounds of PC1 and PC2 of the ISA. This allowed the authors to generate new series that lie outside the boundaries of the original instance space. For each of the target points, the GA searched for an instance individually and started with an initial population consisting of randomly generated instances and existing instances not too close at the target point (to avoid instance replications). To evolve an instance, the GA modified the raw data of the instance. Since the GA was not able to adjust the length of a time series itself, the process was run several times for different time series lengths. The authors observed that despite the equal distribution of the target points in the instance space, there are still large unoccupied regions after incorporating the new instances. The implication is that the time series have natural boundaries within the two-dimensional instance space due to constraints on feature combinations. However, Kang et al. (2017) also reported that their evolved instances are more evenly distributed in the instance space and that new regions are being explored.

To generate instances for machine learning classification, Muñoz et al. (2018) employed a Gaussian mixture model to minimize the mean squared error between the features of a new instance and a target feature vectors (determined by a Latin hyper-cube sample in the two-dimensional instance space with given bounds). Although using a Gaussian mixture model has several advantages, it is not appropriate for generating machine scheduling instances.

Bowly et al. (2020) developed a very powerful CIG for linear programming instances that, when well attributed, achieved a much higher coverage of feature space projections (works like the instance space projection but without dimension reduction and presents two features in a two-dimensional space) compared to a naive generator. To fill gaps in some of the feature spaces, the authors defined a single target feature point and used a local search (with specific instance search operators) to obtain corresponding instances by modifying existing instances from the naive generator and by adjusting the attributes of their CIG. Their

results showed that the approach was able to generate instances filling some gaps. However, even the local search was not able to produce a uniform distribution of instances across all feature spaces.

For the continuous black-box optimization problem, Muñoz & Smith-Miles (2020) presented a Genetic programming approach that treats instance generation as a task similar to a symbolic regression problem. Two strategies were used to generate new instances: The first used a Latin Hypercube Design (LHD) to define target points in the high-dimensional feature space (with eight features), while the second used an LHD in the two-dimensional instance space. They reported that the first strategy was not able to push the boundaries of existing benchmarks very far. However, the second strategy with the target points in the instance space, was able to reach new regions and thus to generate instances with more diverse characteristics.

Smith-Miles et al. (2021) proposed several methods to generate instances near target points in the instance space for the 0-1 knapsack problem. Here, two types of target points were defined: manually placed target points within the interior of the instance space to address gaps and target points on the empirical boundary (represented by the convex hull of the projected hyper-cube of minimum and maximum feature values pruned by unlikely vertices) of the instance space. In this way, a total of 26 target points was defined. To generate weakly structured instances, a GA was implemented to evolve the raw data of existing instances closest to the desired target point. The final population from a single GA run was then used to create 10 new instances that most closely matched the desired target point. To generate strongly structured instances, four different CIGs (with different attributes) were used. Starting with randomly created attribute vectors, a GA evolved these vectors to minimize the distance to the current desired target point. Using the fittest attribute vector, 10 instances were generated for each target point and CIG. The projection of the newly generated instances into the original instance space showed that the structured approach was able to expand the coverage area to some extent, but this approach was not very good at producing diverse instances. The unstructured approach was not able to generate instances in new regions of the instance space.

The only paper that addressed a machine scheduling problem, i.e., the JSP, is that of Vela et al. (2021). The authors used a "Unified Particle Swarm Optimization" (UPSO) algorithm to generate new instance raw data. The primary focus of this work was to generate instances that are easy to solve for one of the four solution methods considered, with the intention that such instances may be hard to solve for the other solution methods. Despite such instances, which are not the focus of this work, the authors proposed to generate so-called "feature-focused" instances. To do this, the authors analyzed five features of the generated JSP instances and used their minimum and maximum values combined with a normalization constant to define target feature values. UPSO was then used to generate instances to match the target feature values. Because the latter approach was studied only with a fixed number of jobs (3) and machines (4) and the discussion of the results is very specific, insights regarding our research are limited.

Coster et al. (2022) proposed an approach to generate instances for the curriculum-based course timetabling problem. To generate new instances, the authors extended the CIG proposed by Lopes & Smith-Miles (2013) with 16 new attributes to have more influence on the generation process. This CIG was then run with attribute values which are sampled from distributions estimated based on real-world instances.

Since several of the used features are correlated, sampling directly from the distributions of the features would lead to unrealistic and often trivially infeasible instances. Therefore, PCA is used for dimension reduction, and the distributions of the transformed parameters were estimated using kernel density estimation. Sampling by these distributions resulted in new principal components, which in turn were reverse transformed into a new set of generation attributes. Finally, infeasible instances (e.g., with a total number of lectures greater than the total number of scheduling options) are filtered out by simple checks. This process was performed iteratively to fill sparse regions of the instance. The authors report that their approach covers the space occupied by real-world instances better than previous generators, and that many of the new instances have characteristics like the real-world instances (in terms of the features analyzed).

For the two-dimensional bin packing problem, Liu et al. (2023) used two GAs to generate instances at target points filling the entire region inside the estimated boundary within the instance space. The boundary is estimated by the "Correlated Limits of the Instance Space's Theoretical or Experimental Regions" procedure. The first GA evolved attributes for an adapted CIG that bases on 2DCPackGen proposed by Silva et al. (2014). This GA was run for each target point. To obtain instances near the target points in the instance space, the second GA evolved parameters (i.e., the widths and heights of the small bins to be placed) that were directly used to compute instance raw data. They report that the first GA was able to effectively fill gaps between existing instances but lacked diversity. However, both approaches were able to generate instances that covered the instance space and expanded beyond the original instances in the center of the instance space.

van Bulck & Goossens (2023) developed a method that uses a CIG to generate new instances for sports timetabling. To do this, they defined a "target instance space" (by the convex hull of the projections of the inequalities that define the bounding box in the high-dimensional feature space) where the target points representing desired feasible real-world-like instances must lie. The target points are then determined by a grid covering the entire target instance space (the number of grid points within the target instance space defines the number of instances to generate). To transform each target point into the corresponding high-dimensional feature vector used by the CIG to generate new instances, the authors proposed a mixed-integer linear program (MILP), since the inverse transformation of the PCA did not lead to suitable results. The projection of the 45 new instances onto the original instance space showed a remarkably good covering of target points by the projected new instances.

Analyzing these most relevant AIGs, all except van Bulck & Goossens (2023), use some kind of (iterative) search method, the so-called meta-generation method, to generate instances that either possess specified target properties (features) or that hit specified target points T^2 in the instance space (after projecting their features). The methods used to generate the corresponding instances ("Modifying raw data", "New raw data", and "CIG with attribute adaptation") vary and no clear superior one can be identified. However, all these papers influenced the development process of our instance generation approach described below.

3 Instance generation by reverse engineering (RIG)

The central aspect that differentiates our approach from the literature is not to search for new diverse,

reasonable, and feasible instances but to generate them by reverse engineering from ISA. Reverse engineering means that we take advantage of the ability of most dimension reduction techniques (e.g., PCA) to perform a reverse projection from the lower dimensional space (here the instance space) to the higher dimensional space (here the feature space). van Bulck & Goossens (2023) claimed in their instance generation approach that such a "simple inverse transformation […] does not suffice to derive a set of high dimensional feature vectors projected at these coordinates" (van Bulck & Goossens, 2023). The reasons given were that relationships between features are ignored by this inverse transformation, and that due to the linearity of the transformation, most feature values are likely to be zero or not within the bounds of the high-dimensional feature space. However, we believe that these problems are directly related to the feature engineering and that features with direct relationships (e.g., two features that are mutually exclusive) should generally be avoided (as is common in machine learning). Regarding the second argument, that the resulting feature vector values are not suitable, we can report that we observed zero feature values extremely rarely. Maybe the unsuitable results of the inverse transformation observed by van Bulck & Goossens (2023) are due to the integer features whereas most of our features are real values. Furthermore, extremely low or high feature values should not be seen as problematic but rather es beneficial for generating a highly diverse set of instances. Of course, the requirement to compute reasonable and feasible instances must be met (as is done by van Bulck & Goossens, 2023, within their feature vector determination model).

Figure 1 illustrates the transformation processes applied in our RIG method:

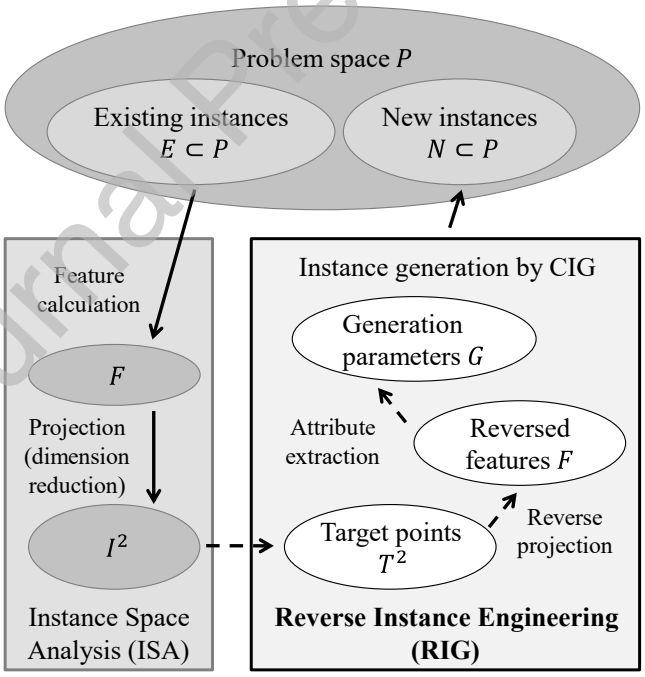

Figure 1: Main course of action of the Reverse Instance Engineering

The first part of our proposed RIG approach is identical to the basic ISA process, with the transformation of instances to points in $\mathbb{R}^2$: $E \xrightarrow{f} F \xrightarrow{p} I^2$.

To obtain a set of desirable feature vectors for generating instances, we use the same approach as used by many other authors and use a function b to determine target points t in the instance space: $I^2 \xrightarrow{b} T^2$. Based on this set of target points T^2, we perform an inverse projection (p^{-1}) to calculate new feature vectors

$p^{-1}(t) \in F$ for each target point $t \in T^2$: $T^2 \xrightarrow{p^{-1}} F$. For computing reasonable and feasible instances, we propose to use the procedures also used by CIGs and influence them with a parameter vector derived by a mapping g from the "reverse engineered" feature vectors: $g(p^{-1}(t)) \in G$, with G representing the set of all parameters required for generating reasonable and feasible instances: $F \xrightarrow{g} G$. We propose to use CIG procedures to generate machine scheduling instances because they offer several advantages over other approaches in the literature: First, CIG procedures can be designed to generate only feasible and reasonable instances. Second, expert knowledge can be integrated (e.g., to obtain instances that resemble the real world). Third, a large and controllable variety of instances can be generated, provided they have the "right" attributes. The advantage of using properly attributed CIG procedures is for example demonstrated by Jooken et al. (2022). The authors use a CIG to generate instances for the 0-1 knapsack problem that are harder to solve than the (evolved) instances proposed in Smith-Miles et al. (2021). The CIG itself represents a transformation from parameters to a new set of problem instances: $G \xrightarrow{CIG} N \subset P$.

To generate the desired diverse, reasonable, and feasible instances, we developed a comprehensive iterative process (a similar process in the context of an ASP is presented in Muñoz & Smith-Miles, 2020) consisting of the six steps illustrated in Figure 2:

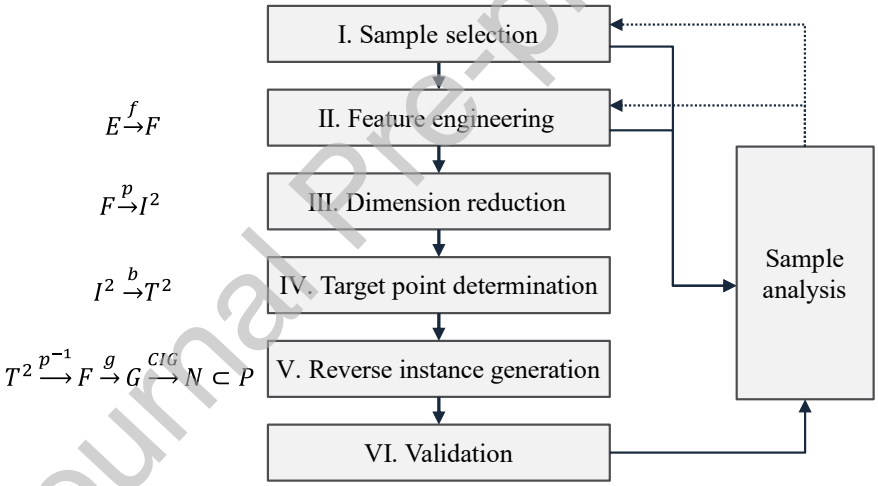

Figure 2: The process steps of instance generation by reverse engineering

The process steps I., II., and VI. are accompanied by a so-called "Sample analysis" (SA) which provides statistical information about a (sub)set of instances (sample) and supports the decision making throughout the iterative RIG process by enabling an effective and traceable comparison of two or more instance samples. For example, if the SA in the validation step shows an unsatisfying diversity of an important instance characteristic (feature), another set of features should be investigated, and the subsequent process steps are repeated. The SA combines elements from descriptive statistics (i.e., measures of central tendency such as mean or median and measures of variability such as standard deviation, variance, or skewness) with statistical graphics such as box plots or violin plots from the area of exploratory data analysis. These elements of the sample analysis are used to describe and summarize the characteristics of an instance sample based on a given set of features $F^{SA} \subseteq F$ that have been determined specifically for the purpose of sample analysis (see Section 3.2).

3.1 I. Sample selection

The purpose of the sample selection step is to define a base sample of instances $E^B \subset E$ from all existing real-world and synthetic instances $E \subset P$. The selection of the base sample, together with the features used, is responsible for the shape of the resulting instance space I^2, and thus in turn for the target space T^2 and the new instances. A natural approach would be to use all existing instances so that $E^B = E$. For some (new) problems, the number of available instances may be very small. In this case, E could be extended with existing (adapted) instance generation procedures (as for example done by Coster et al., 2022). Furthermore, for some problems, authors might have published very specific synthetic instances or instances of a very distinct real-world case are part of E. In these cases, it may be appropriate not to use all instances of E but only a subset $E^B \subset E$. A good example for this is the SMTWTP. There are three instance sets (wt40, wt50, and wt100) provided by the OR-Library (Beasley, 1990) and a CIG, which is commonly used in the literature (see e.g., Gahm et al., 2019) and was also used in this study to generate six instance sets (gkt25, gkt50, gkt100, gkt200, gkt400, and gkt800). The sample set combining all the instances is named E^{STD}. The same CIG approach, but with different attributes, was used by Geiger (2010) to generate two new instance sets (geigS and geigP) to be more challenging for state-of-the-art algorithms. The sample set combining these two sets and E^{STD} is called E^{STD+G}.

The effect of the instance sets provided by Geiger (2010) on the resulting instance space is illustrated in Figure 3.

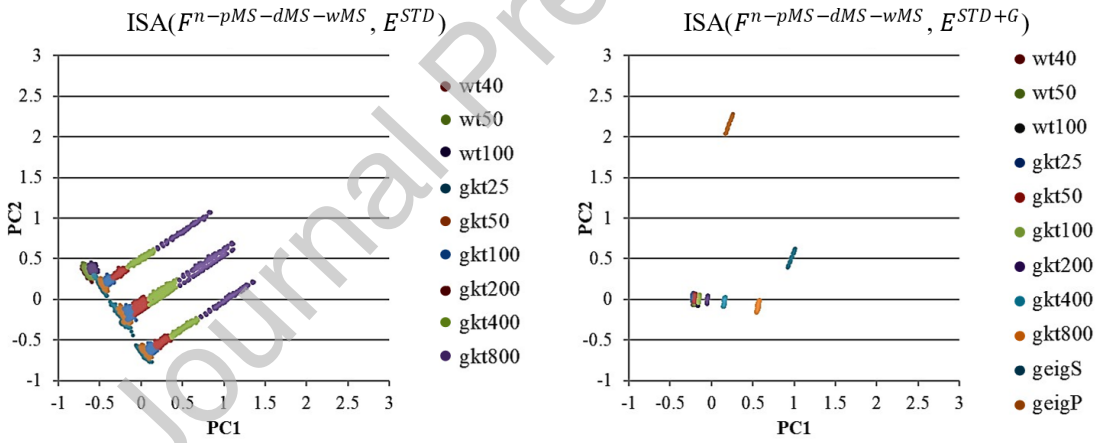

Figure 3: Illustration of the effect of different base samples on the resulting instance space (with identical feature set)

As can be seen on the right-hand side of Figure 3, the instance space is very different when the instance sets geigS and geigP are part of the base sample.

The decision on the final composition of the base sample E^B may depend on the specific research project and whether all available instances in E are considered to be reasonable. It is also possible that after validating the newly generated instances, the conclusion is to adapt the base sample and start the iterative process again (see Figure 2).

Some authors recommend eliminating instances with outliers from the instance samples or performing a log transformation on all features to prevent the instance space from being unduly distorted by outliers (cf., e.g., Smith-Miles & Bowly, 2015 or Lechien et al., 2023). Since the purpose of the ISA and the

subsequent instance generation is to provide as diverse a set of instances as possible, we do not recommend outlier elimination of any kind, as these "special" instances may provide the information needed to generate the most interesting instances at the boundaries of the instance space. We also do not use log transformations to allow visual identification of "special" instances in the instance space (cf. purpose P1 and P3).

3.2 II. Feature engineering

The term “feature engineering” is commonly used in machine learning to describe the process of defining and computing numerical values (features) to represent a given object. It is an iterative process and usually involves a considerable amount of trial and error to get the best features for the intended purpose. In the context of machine scheduling and ISA/RIG, given objects are problem instances and their features, combined in a feature vector, should comprehensively describe their characteristics. Like the definition of the base sample E^B, it may also be advantageous not to use all conceivable features from the feature space F, but to select a subset s of features $F^s \subseteq F$ that contains features that represent the (most) important characteristics of an instance and that provides the required information to derive appropriate attributes for the CIG procedure. Accordingly, we first need to determine F and then identify a most suitable feature set from the power set of F: $F^s \in P(F)$.

In their approach to measure instance difficulty, Smith-Miles & Lopes (2012) propose to use problem independent features based on “Fitness landscape analysis”. This analysis is based on computed solutions characterizing the search space of applied solution methods. However, as “[…] the link between problem difficulty and fitness landscape structure is poorly understood.” (Bierwirth et al., 2004) and our approach is designed to be completely independent of solution methods, we do not recommend using such features within RIG. For the same reason (being independent of solution methods and their performances), we are not applying the SIFTED algorithm to identify features as proposed in Smith-Miles & Muñoz (2023). Instead, we rely on the expertise of scheduling researchers to identify suitable features and the proposed iterative process (see Figure 2). With respect to machine scheduling, three starting points for determining F based on literature analyses can be recommended: First, the control attributes of existing CIGs as these are usually used to control the most important instance characteristics from the perspective of the respective authors. Second, most empirical analyses investigate the performance of solution methods with respect to instance characteristics. Third, whenever machine learning is applied (e.g., in the context of the ASP), features describing instances have already been determined.

In general, we distinguish between the features used in the sample analysis ($F^{SA} \subseteq F$) and those used for reverse engineering. In the former case, the number of features can be arbitrarily large (however, statistical graphics and summary tables may become illegible if it is too large). In the latter case, the number of features in the feature set should be as small as possible and as large as necessary. As large as necessary means that all the main instance characteristics should be represented (to allow for an appropriate ISA) and that the necessary information to derive the required parameters for the CIG procedures is provided. The number of features should be as small as possible, because a smaller number of features is more likely to result in a high explained variance by the dimension reduction (e.g., in the first two principal components of a PCA) and in higher coefficients needed for the inverse transformation. For these reasons, (strongly)

related features representing the same characteristic should also be avoided (e.g., mean and median of an instance data parameter such as processing time, both describing the central tendency). Whenever two or more features can express the same instance characteristics, we recommend using only one to preserve the information while avoiding unwanted side effects. However, which feature combination complies best with these requirements (and the purpose of reverse instance engineering) is the result of the iterative process illustrated in Figure 2: $E^B \xrightarrow{f} F^B$. Result of the feature engineering is a feature set that contains $|F^B|$ features from F^B and appropriately represents each instance $i \in E^B$ by feature vector $v(i) = (f_i^1, f_i^2, \dots, f_i^{|F^B|})$.

As already stated, within our RIG process, we do not perform log transformations after feature calculation but a min-max normalization to [0, 1] (this ensures that each feature has an equal share in determining the direction of maximal variation; van Bulck & Goossens, 2023). That means, each value f_i^x of feature $x \in F^B$ and instance i is transformed to $\dot{f}_i^x$:

$$\dot{f}_i^x = (f_i^x - \min(f_i^x | i \in E^B)) \,/\, (\max(f_i^x | i \in E^B) - \min(f_i^x | i \in E^B)).$$

The final feature vector of instance i is given by $\dot{v}(i) = (\dot{f}_i^1, \dot{f}_i^2, \dots, \dot{f}_i^{|F^B|})$. Note that any feature transformations must be inverted during the reverse engineering (e.g., after the inverse projection from the target space to the feature space $T^2 \xrightarrow{p^{-1}} F$), and thus, the min-max normalization must also be inverted.

3.3 III. Dimension reduction

For the projection from the high-dimensional feature space to the two-dimensional instance space ($F^B \xrightarrow{p} I^2$), any dimension reduction technique with the possibility of reverse transformation can be used. In the literature, most approaches using dimension reduction for ISA apply PCA as dimension reduction technique (e.g., Smith-Miles & Bowly, 2015, Neuenfeldt Júnior et al., 2019, Weise et al., 2020, Kletzander et al., 2021, Scherer et al., 2024, or Sun et al., 2024). Only Muñoz et al. (2018) and Muñoz et al. (2021) use their developed optimization-based "Prediction Based Linear Dimensionality Reduction – PBLDR" method and Smith-Miles et al. (2021) use the "Projecting Instances with Linearly Observable Trends (PILOT)" method that is identical to PBLDR. Smith-Miles & Muñoz (2023) also use PILOT to encourage linear trends in features and solution method performance. Both methods, PCA and PILOT, generally provide the possibility of inverse transformations. However, since the aspect of solution method performance is explicitly not considered, we are not applying PILOT but PCA for dimension reduction. Another advantage is its high efficiency (also with large datasets) and its availability in many software packages. Therefore, we use it for the dimension reduction here: $F^B \xrightarrow{PCA} I^2$.

Before applying the PCA, the feature values must be mean centered to ensure that a feature vector is projected to the origin of the two-dimensional instance space. Accordingly, we transform the feature values of $\dot{v}(i)$ by $\ddot{f}_i^x = \dot{f}_i^x * \text{mean}(f_i^x | i \in E^B)$ and obtain $\ddot{v}(i) = (\ddot{f}_i^1, \ddot{f}_i^2, \dots, \ddot{f}_i^{|F^B|})$. Most PCA implementations perform this transformation automatically, but since the mean centering must also be inverted during the reverse engineering, we highlight this step here.

Since the desired instance space I^2 consists of two dimensions, we are only interested in the two principal components with the largest explained variance (eigenvalues) to define a so-called sample point p ($\mathbb{R}^2$) for an instance i: $p_i(\mathrm{PC1}_i, \mathrm{PC2}_i) = PCA(\ddot{v}(i))$. All the sample points of the instances of a sample constitute the instance space as shown in Figure 3 (right-hand side). Note that in all the instance space graphs in this paper, the x-axis (abscissa) corresponds to PC1 and the y-axis (ordinate) corresponds to PC2.

The basis of the transformation (and the inverse transformation) are the projection coefficients (weights) that map the feature vectors to the two principal components. Figure 4 shows the projection coefficients computed by the PCA for the two feature sets $F^{n-pMS-dMS-wMS}$ and $F^{n-pMS-dRT-wMS}$ (see Section 4.1.2 for more details). The total explained variance is 91.6% (61.4% for PC1 and 30.2% for PC2) and 78.3% (50.5% for PC1 and 27.8% for PC2), respectively. Besides the higher explained variance, the projection by $F^{n-pMS-dMS-wMS}$ is more stable and more meaningful because each feature (and thus the important instance characteristics) is represented in at least one component by a sufficiently high coefficient.

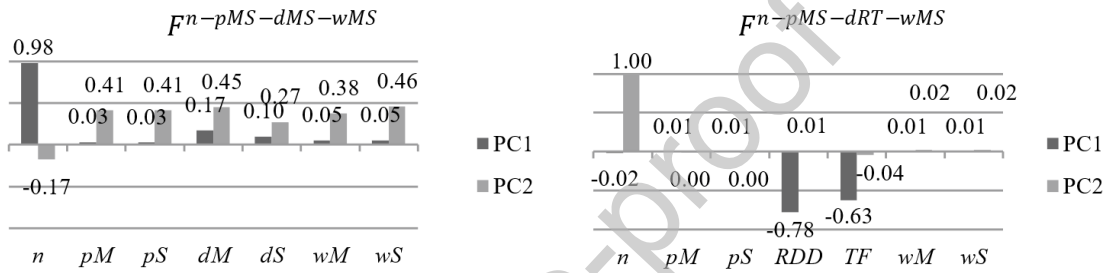

Figure 4: Projection coefficients for different feature sets (with sample set E^{STD+G}).

In general, both aspects, a high explained variance and a suitable representation of each feature by projection coefficients, are necessary conditions for reverse engineering to be possible. Therefore, with respect to the example, we conclude that reverse engineering is possible for the SMTWTP with the feature set $F^{n-pMS-dMS-wMS}$ and sample E^{STD+G}.

3.4 1V. Target point determination

To generate a new set of diverse and reasonable instances, we follow the approach often successfully used in the literature (e.g., Smith-Miles & Bowly, 2015) and define a set of target points with the instance space: $T^2 \subset I^2$. To avoid generating unreasonable instances, it is common to first define a target instance space (a subspace of the instance space) that contains points that are most likely to lead to reasonable instances. Two approaches are used for this: The first approach is based on (empirical) bounds of feature values defining vertices of a hyper-cube (sometimes combined with a pruning of features due to correlations between them; cf., e.g., Smith-Miles et al., 2021 or Liu et al., 2023) and the projection of the (pruned) hyper-cube to the two-dimensional instance space (cf., e.g., Smith-Miles & Bowly, 2015 or van Bulck & Goossens, 2023). In the second approach, the target instance space is determined directly by the instance space, i.e. by the minimum and maximum values of the two principal components adjusted by given values to not only filling "holes" in the instance space but also to additionally enlarge the overall space that is occupied by the existing instances (to push the boundaries of the instance space; cf. e.g., Kang et al., 2017 or Muñoz & Smith-Miles, 2020). Since we can assume that all features are well represented by the two principal components (see previous sections), and since reasonableness and feasibility are ensured by the subsequent steps of the CIG

application, we follow the second approach and define the target instance space directly based on the instance space. However, instead of defining the boundaries of the target instance space by adjusting the minimum and maximum component values, we compute the rotated minimum bounding rectangle ($rMBR$; rotated means that the edges of the rectangle are not necessarily parallel to the axes of the instance space). In this way, we can better consider the "shape" of the existing instances in the instance space. The result of the $rMBR$ calculation is a set of four points: $rMBR = \{p_r(x_r, y_r) | r = 1, 2, 3, 4\}$ (numbering is clockwise from the top left). To increase the diversity of the new instances, we enlarge this $rMBR$ by a relative factor δ, which is used to calculate two adjustment offsets related to the maximum deviations of the $rMBR$ points to the origins of the two principal components: $\Delta x = \delta \cdot \max\{|x_r|\}$ and $\Delta y = \delta \cdot \max\{|y|\}$. In this way, we consider the magnitude of both components and amplify them accordingly. The resulting adjusted $rMBR$ is defined as follows:

$$rMBR = \{p_1(x_1 - \Delta x, y_1 + \Delta y),\ p_2(x_2 + \Delta x, y_2 + \Delta y),$$
$$p_3(x_3 + \Delta x, y_3 - \Delta y),\ p_4(x_4 - \Delta x, y_4 - \Delta y)\}.$$

However, any other approach to defining the boundaries of the target instance space would be applicable.

To finally determine the set of target points T^2, we use the parameter π to define the number of points on each axis of the $rMBR$, thus expanding a grid consisting of π^2 target points. Both parameters can be used to control the exploration (by a high δ value) and the exploitation (by a high π value) of the instance space. Figure 5 shows the $rMBR$ and the determined target points T^2 for the SMTWTP example.

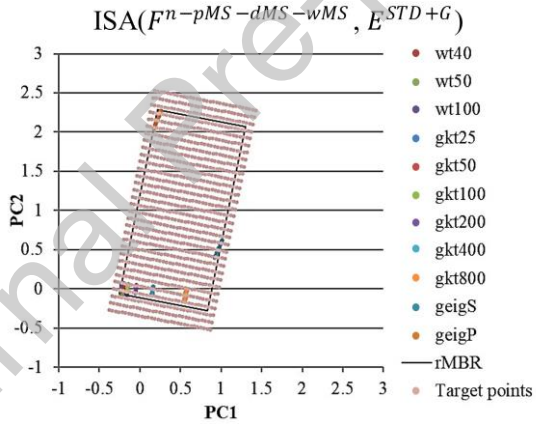

Figure 5: Illustration of *rMBR* and target points T^2 (with δ=0.1 and π=32)

Summarizing this process step, we use a function b with the two parameters δ and π to determine the target points based on the instance space given by the previous steps: $I^2 \xrightarrow{b(\delta,\pi)} T^2$.

3.5 V. Reverse instance generation

First sub-step of the reverse instance generation itself is the inverse projection of the target points onto the feature space $T^2 \xrightarrow{p^{-1}} F$. The coordinates of a target point t correspond to the two principal components $(p_t(\mathrm{PC1}_t, \mathrm{PC2}_t))$ and the mean centered feature vector $\ddot{v}(t) = (\ddot{f}_t^1, \ddot{f}_t^2, \dots, \ddot{f}_t^{|F^B|})$ results from the inverse transformation $I^2 \xrightarrow{PCA^{-1}} F^B$: $\ddot{v}(t) = PCA^{-1}(p_t(\mathrm{PC1}_t, \mathrm{PC2}_t))$.

Inverting the mean centered features $(\dot{f}_t^x = \ddot{f}_t^x / \mathrm{mean}(f_i^x | i \in E^B))$ leads to the scaled feature vector $\dot{v}(t) = (\dot{f}_t^1, \dot{f}_t^2, \dots, \dot{f}_t^{|F^B|})$, which in turn is inverted by the original scales to the new feature vector

$v(t) = (f_t^1, f_t^2, \dots, f_t^{|F^B|})$ (with $f_t^x = \dot{f}_t^x \cdot (\max(f_i^x | i \in E^B) - \min(f_i^x | i \in E^B) + \min(f_i^x | i \in E^B))$).

In the next step, the parameters for the CIG procedure must be determined individually for each target point and its feature vector $v(t)$: $g(v(t)) \in G$. Remember that we differentiate between the control attributes of a CIG and the parameters used by the CIG procedure. For example, the CIG used by Gahm et al. (2019) for the SMTWTP has the attribute "processing time variability" with the manifestations "low" and "high". The attribute is in turn used to define the parameters "lower bound" and "upper bound" of the processing time interval from which the processing times are finally drawn from: "low" results in [45, 55], while "high" results in [1, 100]. Because of the loss of information by mapping one or more feature values to existing CIG attribute manifestations (the resulting instance would most likely be nowhere near the target point), and because newly introduced attribute manifestations may not be processable by the CIG, we propose to map features directly to parameters used by the CIG procedure. In the example, this would mean that the interval bounds are directly derived from one or more feature values (e.g., by the mean and the standard deviation of the processing time). However, sometimes the determination of CIG attributes may be unavoidable. In this case, we recommend deriving threshold values from the base samples to determine the attribute based on the feature values. For example, if the standard deviation of the processing time is below the threshold value, the attribute is set to "low", otherwise it is set to "high".

While deriving attributes and parameters from reversed features, we can also integrate mechanisms that ensure feasible and reasonable instances. For example, the number of jobs should be greater than or equal to one, and due dates should be greater than or equal to zero. Whenever a lower or upper bound on the generation parameters must be respected, there are two options: adjust the parameter to the corresponding bound (e.g., set the number of jobs to a given minimum value) or discard the instance completely. The advantage of the first option is that exactly the desired number of instances is computed at the cost of diversity, as instances generated with the same parameters will result in "clusters" of very similar instances. Therefore, we recommend using the option to discard instances, with the advantage that the diversity is kept on a higher level and "clusters" of instances (e.g., at the borders of the $rMBR$) could be avoided (at the cost of an unpredictable number of instances). The discarding of nearly identical instances to avoid biases is also proposed by Alipour et al. (2023). If the resulting number of instances is not sufficient (e.g., for statistical analyses), the target point determination parameters δ and π can be adjusted. Figure 6 illustrates the difference between both options for the SMTWTP.

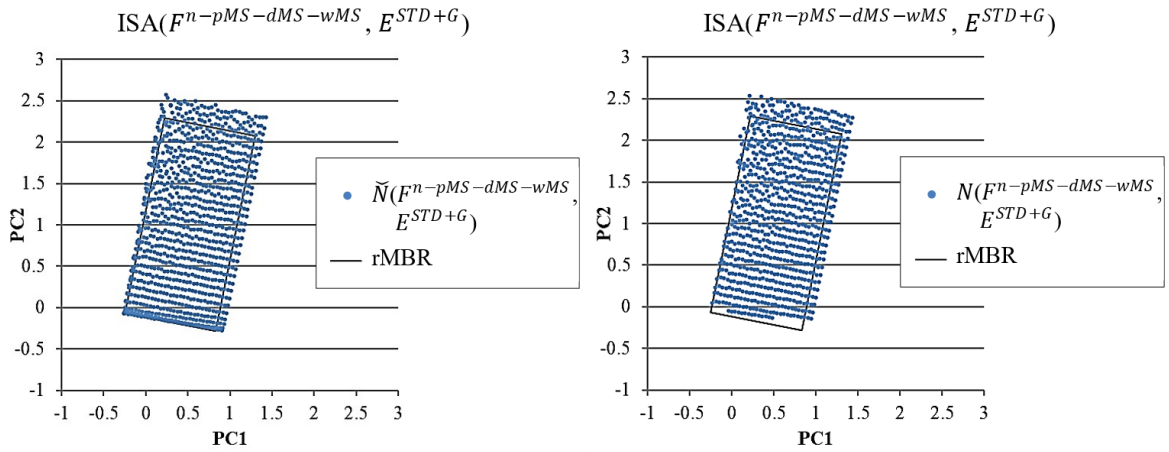

Figure 6: Comparison of instances with generation parameter adjustment ($\breve{N}$, left side) and instance discarding (N, right side) projected to the same instance space.

In Figure 6, on the left side, where all desired $|\breve{N}| = \pi^2 = 1{,}024$ instances are projected, some clustering (due to parameter adjustments) on the left border and at the bottom border of the $rMBR$ can be observed. On the right side, where $|N| = 863$ instances are projected, this clustering is avoided by discarding instances. The specific adjustment and discarding mechanisms will be presented in the sections related to the specific scheduling problems.

In summary, executing the CIG procedure with the derived attributes and parameters yields a new set of instances $N \subset P$. For simplicity, and because we recommend using the discarding option, we use N to indicate a set of generated instances where unreasonable instances are discarded in the following.

3.6 VI. Validation

To validate that the newly generated instances are useful for the purpose of performance testing, three steps should be performed: First, the new instance set N is projected onto the instance space obtained with the feature set F^B, the original base sample E^B (i.e., without N), and the corresponding projection coefficients. Figure 7 illustrates this "re-projection" on the right side.

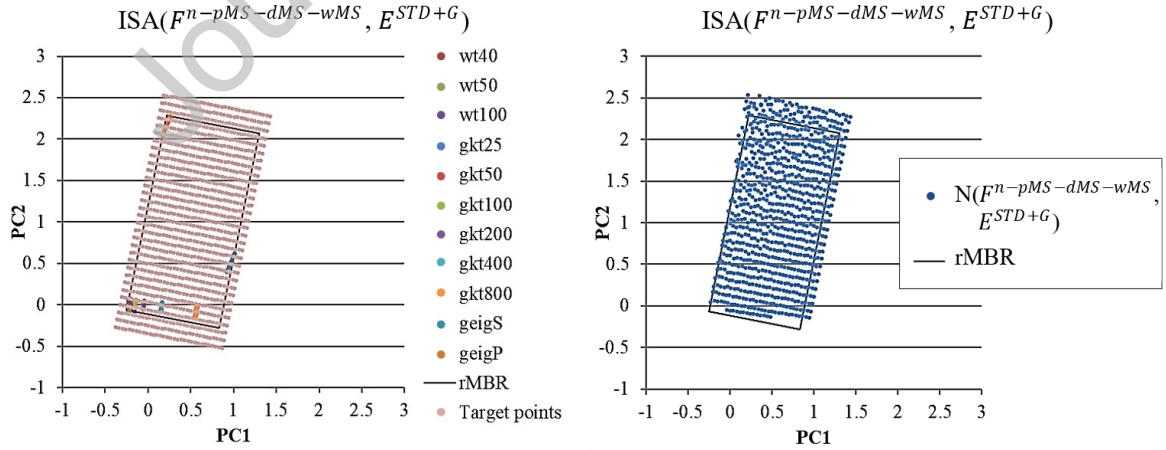

Figure 7: Comparison of target points (left side) and new instances (right side) projected to the same instance space.

As can be seen by comparing the left and right sides of Figure 7, the newly created instances generally fill the area of the instance space as desired by the determined target points (i.e., they fill the gaps and explore new regions).

It can also be seen that the target points left of and/or below the $rMBR$ are not "hit" and instances are missing. This is because instances are discarded based on specified bounds to ensure reasonable instances (e.g., a minimum number of jobs per instance or a minimum processing time of one). This is an example of how expert knowledge can augment the CIG to generate reasonable instances. Hereby, plotting feature distributions into the instance space is helpful to identify relationships between areas in the instance space and specific features (cf., Figure 20 in Appendix A3).

The second validation step is to repeat the ISA with the combined instance set $E^B \cup N$ and feature set F^B. As with the ISA with E^B and F^B, the projection should result in a more or less "rectangular" shape (see left side of Figure 8).

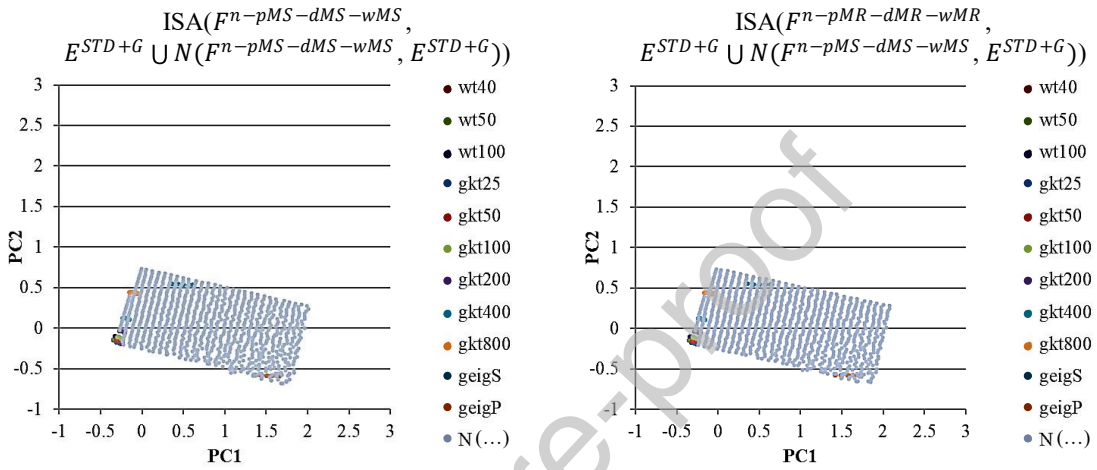

Figure 8: Comparison of ISAs with different feature sets for projecting the new (combined) sample ($E^B \cup N$).

Note that the location of the new instances in the instance space is different to the ISA shown in Figure 7, as the base sample is now extended by the new instances. Such an analysis with an updated instance space has also been proposed in Smith-Miles et al. (2021). In addition, an ISA with the new base sample and a different set of features (also with a high explained variance) could be performed (see right side of Figure 8). Again, the projection should result in a more or less "rectangular" shape. Note that the "quality" of the second ISA is obviously worse than the first one with the "original" feature set. Note that the "quality" of the second ISA is obviously worse than the first one with the "original" feature set.

The third step of the validation involves comparing the SAs with the feature set F^{SA} and the sample sets E^B and $N(F^{n-pMS-dMS-wMS}, E^{STD++G})$, respectively. Box plots or violin plots (see Figure 9) are useful for evaluating whether the desired diversity has been achieved.

As discussed before, the whole process is an iterative one and when the validation results are not satisfying, steps I. to V. must be adapted and repeated.

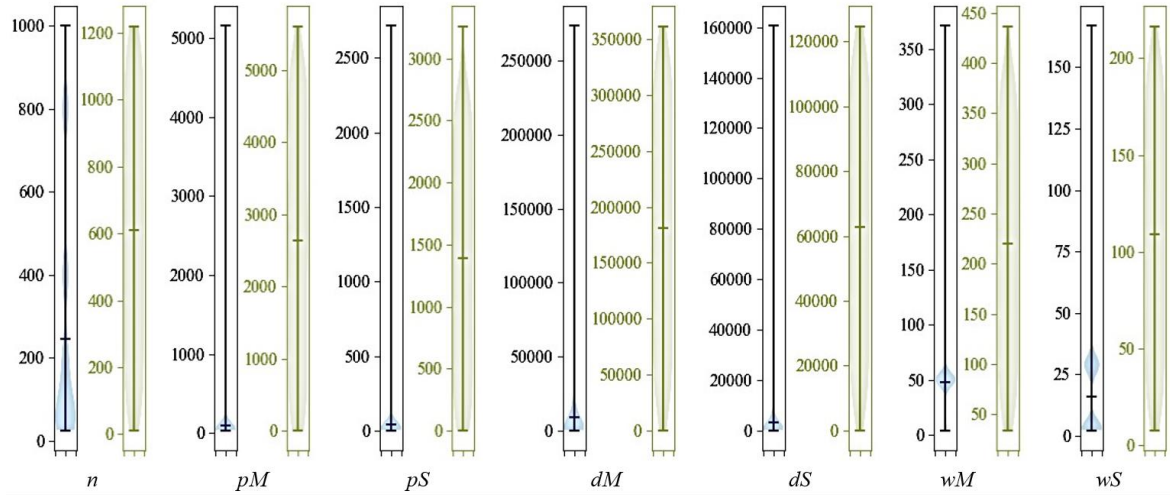

Figure 9: Violin plots of selected features for the original base sample E^{STD+G} (blue) and for the new sample set $N(F^{n-pMS-dMS-wMS}, E^{STD+G})$ (green).

4 Evaluation

To evaluate the applicability of our RIG approach to generate new diverse, feasible, and reasonable machine scheduling instances, we applied it to generate instances for three different problems: the single machine weighted tardiness problem (SMTWTP), the job shop scheduling problem (JSSP), and the parallel serial-batch scheduling problem with incompatible job families (PSBIJF). Note that all instances used as base samples, as well as the newly created instances, are made available for download at Mendeley Data (Gahm, 2025). For easy integration into new projects, we provide all instances in a unified format.

4.1 Single-machine weighted tardiness problem

The SMTWTP is a classic problem in machine scheduling theory. It focuses on scheduling a set of n jobs with processing times (p), priority weights (w), and due dates (d) per job on a single machine with the objective of minimizing the total weighted tardiness of all jobs. According to the three-field notation of Graham et al. (1979), we address the $1||\sum w_j T_j$ – problem.

4.1.1 Sample selection

The base sample consists of a total of 3,425 instances. 375 test instances are derived from the OR-Library from Beasley (1990) and consist of 125 instances for the problem sizes (number of jobs) $n \in \{40, 50, 100\}$. These three sample sets are named wt40, wt50 and wt100. The next six sample sets (gkt25, gkt50, gkt100, gkt200, gkt400, and gkt800), each consisting of 500 instances, are generated based on the procedure proposed by Gahm et al. (2019) and differ by the number of jobs $n \in \{25, 50, 100, 200, 400, 800\}$ (amongst other attributes). Other generation attributes are identical for each set. For example, the processing times and the priority weights are drawn from the two uniform distributions [45, 55] and [1, 100] (to represent low and high variability). These nine sample sets form the first base sample called E^{STD}. Two additional sample sets are provided by Geiger (2010). The sample set *geigS* consists of 25 test instances with n=1000, processing times drawn from the uniform distribution [1, 1000], and weights drawn from the uniform distribution [1, 100]. The sample set *geigP* also consists of 25 test instances computed by drawing processing times and weights from a set of prime numbers ensuring unique combinations of weight and processing time

values. Combining these two sample sets with E^{STD} leads to a second base sample called E^{STD+G}. Common to all instances in all sample sets is that the due dates are drawn equally from an interval defined by the two attributes "tardiness tightness" (tf) and "due date range" (rdd): $[P(1-tf-rdd/2), P(1-tf+rdd/2)]$ (with P :=total processing time).

The influence of the two base samples on the ISA is illustrated by Figure 3. Since we want to achieve a large variety of instances, we will use E^{STD+G} as the base sample in the following.

4.1.2 Feature engineering

From the instance generation procedures found in the literature, we have derived the following main characteristics of SMTWTP instances: the number of jobs and the central tendency and variability of the processing times, of the weights, and of the due dates.

Table 2: Feature space *F* for the SMTWTP

Abbreviations	Features
n	Number of jobs
pM, pS, pCV, pSK, pR	Statistics (mean, standard deviation, coefficient of variation, skewness, and range) for the processing times of all jobs
wM, wS, wCV, wSK, wR	Statistics for the weights of all jobs
dM, dS, dCV, dSK, dR	Statistics for the due dates of all jobs
RDD	Due date range: $(\max(dJ) - \min(dJ))/\max(dJ)$
TF	Due date tightness: $1 - dM/P$ (with P :=total processing time)

To represent these characteristics, the features in Table 2 are defined. For the SA, we use the feature set $F^{SA} = F$. To identify potential feature subsets for the reverse instance engineering for the SMTWTP, we use the data provided by the SA with F^{SA} and E^{STD+G}. The SA revealed that skewness is not important for processing times, weights, and due dates (because of their generation by drawing from uniform distributions) and therefore, these are not part of the feature sets to be investigated in the following (see Table 3).

Table 3: Feature sets for the reverse engineering (SMTWTP)

Feature set	Features	Total explained variance (by PC1, by PC2)
$F^{n-pMS-dRT-wMS}$	*n, pM, pS, RDD, TF, wM, wS*	0.78 (0.50, 0.28)
$F^{n-pMCV-dRT-wMCV}$	*n, pM, pCV, RDD, TF, wMean, wCV*	0.55 (0.30, 0.24)
$F^{n-pMR-dRT-wMR}$	*n, pM, pR, RDD, TF, wM, wR*	0.78 (0.51, 0.28)
$F^{n-pMS-dMS-wMS}$	*n, pM, pS, dM, dS, wM, wS*	0.92 (0.61, 0.30)
$F^{n-pMCV-dMCV-wMCV}$	*n, pM, pCV, dM, dCV, wM, wCV*	0.62 (0.34, 0.28)
$F^{n-pMR-dMR-wMR}$	*n, pM, pR, dM, dR, wM, wR*	0.92 (0.61, 0.30)

4.1.3 Dimension reduction and target point determination

The six feature sets listed in Table 3 are used to perform ISAs and the explained variances obtained are given in the last column. As can be seen, the feature sets $F^{n-pMS-dMS-wMS}$ and $F^{n-pMR-dMR-wMR}$ lead to the same high explained variances and so we could use both for the RIG. Furthermore, the projection coefficients computed by the PCA are identical.

The target points T^2 for reverse engineering are determined with the parameters δ=0.1 and π=32 (cf. Figure 5). This leads to a maximum number of 1,024 new instances.

4.1.4 Reverse instance generation

To generate a new instance $i^t \in N$ for each target point $t \in T^2$, we first compute the inverse transformed feature vector $v(t)$ based on the target point coordinates in I^2 and the feature set $F^{n-pMS-dMS-wMS}$ (also $F^{n-pMR-dMR-wMR}$ would be possible):

$$v(t) = (\hat{n}_t, pM_t, pS_t, dM_t, dS_t, wM_t, wS_t).$$

The next step is to determine the parameters $g(v(t))$ required for the instance generation. Since we are essentially using the instance generation procedure proposed by Gahm et al. (2019) for our reverse engineering, we need to determine the number of jobs n_t and the distribution intervals for processing times (lb_t^p, ub_t^p), weights (lb_t^w, ub_t^w), and due dates (lb_t^d, ub_t^d). To obtain SMTWTP instances with reasonable values, we propose using the following bounds: $LB^n = 10$, $LB^p = 1$, $LB^d = 1$, and $LB^w = 1$.

When following the first option and not discarding unreasonable instances but adjust the generation parameters, the number of jobs for a new instance based on target point t is defined by: $n_t = \max\{LB^n, [\hat{n}_t]\}$ (note that $[x]$ means integer closest to x in this paper). Since we know that the processing times are based on uniform distributions, the lower bound for the processing times is calculated by $lb_t^p = \max\{LB^p, \lfloor pM_t - (\sqrt{12} \cdot pS_t) \cdot 0.5 \rceil\}$ and the upper bound by $ub_t^p = \max\{lb_t^p + 1, \lfloor pM_t + (\sqrt{12} \cdot pS_t) \cdot 0.5 \rceil\}$. The interval bounds for the weights and the due dates are calculated accordingly. For each job of the new instance, the corresponding values are drawn from the calculated intervals with equal distribution.

Selecting the second option and discarding instances would mean using $n_t = [\hat{n}_t]$ and omitting the maximum function when computing the lower bounds. Accordingly, we would only accept instances with $[n_t] \geq LB^n$, $lb_t^p \geq LB^p$, $lb_t^d \geq LB^d$, and $lb_t^w \geq LB^w$. With this approach, 161 instances are discarded, and the new set N contains 863 instances instead of 1.024 instances.

4.1.5 Validation

The validation results already discussed in Section 3.6 show that our reverse engineering process was able to generate instances that are close to the target points and well distributed throughout the instance space. By comparing the sample sets E^{STD+G} and $N(F^{n-pMS-dMS-wMS}, E^{STD+G})$ via the violin plots in Figure 9, we see that, among the new instances, all six job data-related features have better coverage of their respective intervals. Therefore, we can conclude that the new instance set $N(F^{n-pMS-dMS-wMS}, E^{STD+G})$ has a high diversity and therefore forms the final instance set generated by RIG for the SMTWTP. Note that due to the instance generation procedure, we also can guarantee that the instances are feasible and reasonable.

4.1.6 Application of the new instances

In previous sections, we demonstrated that the proposed RIG approach can generate diverse, feasible, and reasonable instances. In this section, we will analyze whether the new instances offer advantages in terms of the four purposes (P1 to P4; see Section 1) associated with solution method performance testing. To demonstrate the advantages of the new instances $N(F^{n-pMS-dMS-wMS}, E^{STD+G})$ compared to the base

sample E^{STD+G}, we will analyze the performance of 15 heuristics applied by Gahm et al. (2019) to solve the $1||\sum w_j T_j$ – problem. These heuristics include the dispatching rule-based heuristics SWPT, WEDD, EHD, WMDD, WCR, WCoverT, ATC, MATC, AR, MAR, BT31WT, QAR, BACK, and QB6, and the new solution method at that time, the decision theory-based heuristic DTS.

First, we analyze whether the new instance set (N) is more challenging than the original base sample (E). To that, on the left of Figure 10, we show the number of solution methods that computed the best solutions in terms of the relative proportion of all instances.

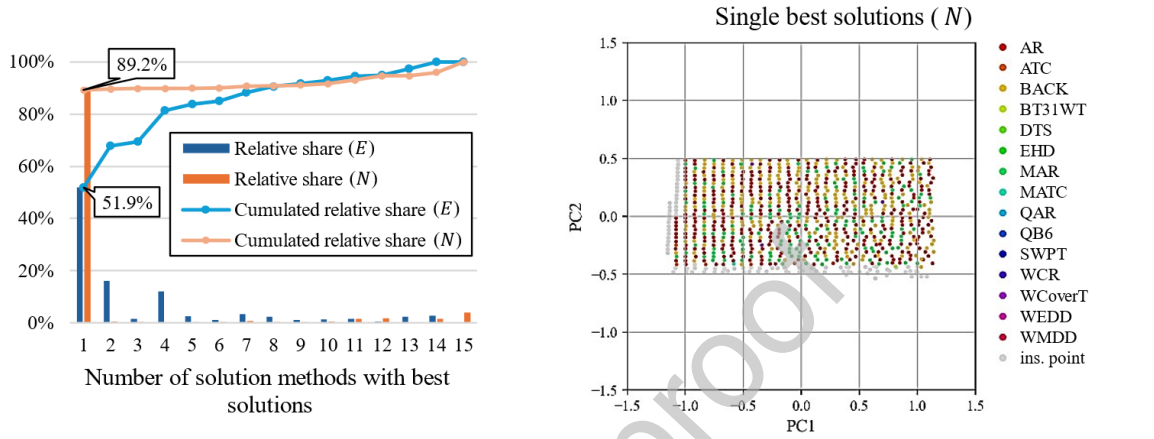

Figure 10: Identification and analysis of challenging instances in the RIG instance set.

As can be seen, the number of instances in which only one solution method computed the best solution is remarkably higher for the new instance set N (89.2%) than for the original base sample E (51.9%). Therefore, since only one solution method can compute best solutions for most of the new instances, we conclude that N provides more challenging instances (Smith-Miles et al., 2014, also used this reasoning to determine challenging instances). Furthermore, the scatter plot on the right side of Figure 10 shows an ISA depicting the solution methods that computed single best solutions for one of the new instances. While no heuristic is superior in a specific region, this plot shows that the three heuristics AR, BACK, and MAR provide unique best solutions for most instances. The plot also shows that instances at the bottom and left of the instance space are easier to solve because more than one heuristic computed the best solution for these instances.

Next, Figure 11 visualizes the solution quality of the AR heuristic (one of the best-performing heuristics for the SMTWTP) and the "new" DTS heuristic using their RIW values ("the relative percentage improvement versus the worst objective value per instance", see Gahm et al., 2019). Note that RIWs are a relative measure for comparing solution methods (purpose P4), not an absolute one, and that larger values are better than smaller ones. Figure 11 presents the results for the combined set of instances $E \cup N$. However, the solitary consideration of the new instances N would provide the same insights.

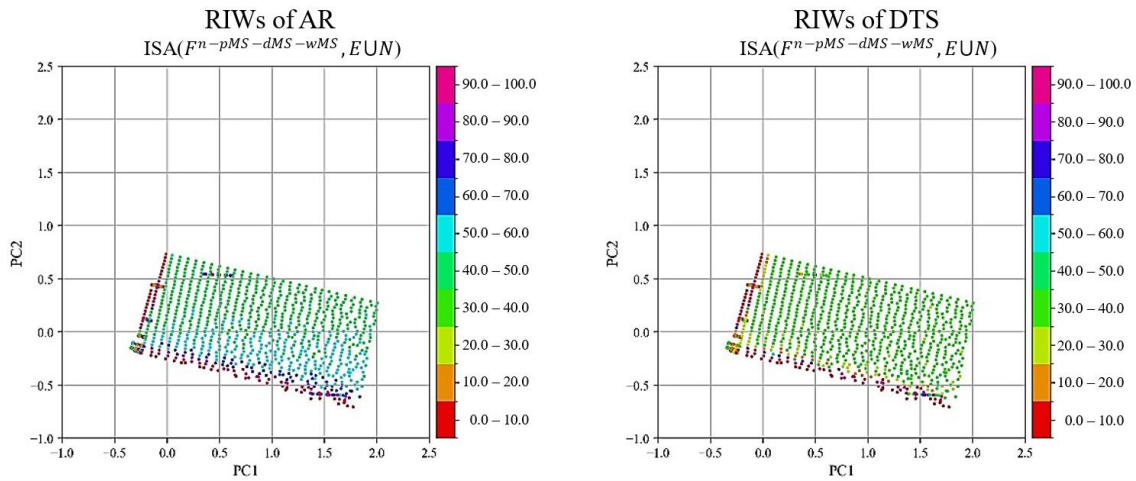

Figure 11: Solution quality of AR and DTS illustrated by the new RIG instances.

Regarding purpose P2, Figure 11 shows that DTS generally achieves good results in the combined instance set (EUN). Regarding purpose P1, however, there are some regions where AR achieves RIWs between 50% and 60%, while DTS achieves only 30% to 50%. Together with the feature projections illustrated in Figure 20 in Appendix A3 purpose P3 is fulfilled.

In summary, the RIG approach is best suited to provide SMTWTP instances for an exhaustive solution method performance testing because it provides diverse, feasible, reasonable, and challenging instances. This is particularly true when the performance analysis framework ISA is applied, but not only.

4.2 Job-shop scheduling problem

The JSSP is a classic NP-complete optimization problem in machine scheduling. It involves determining an efficient way to process a set of n jobs, each consisting of a sequence of operations o with a processing time p_o (normally, the number of tasks is equal to the number of machines), on a set of m machines with specific constraints (e.g., the operations of one job cannot be processed in parallel). The sequence of machines each job is processed on is individual for each job. The goal is to optimize one or more objectives, whereas most often minimizing the maximum completion time of all jobs (makespan) is focused on. According to the three-field notation of Graham et al. (1979), the problem is classified by $J||C_{max}$ (for a more detailed definition see Strassl & Musliu, 2022).

4.2.1 Base samples

The first investigated base sample E^{OR} for the JSSP is based on eight instance sets from the OR-Library (Beasley, 1990): abz (with 5 instances), dmu (80), ft (3), la (40), orb (10), swv (20), ta (80) and yn (4) (the detailed references can be found in van Hoorn, 2018). These sets differ in the number of jobs, the number of machines, and the range from which processing times are uniformly drawn.

In addition to these instances, recently generated instance sets from Strassl & Musliu (2022) are investigated. The authors generated five sets (with number of jobs (n) and number of machines (m) from the set {1, 5, 10, 15, 20, 30, 40, 50, 60, 70, 80, 90, 100}) that differ in the way the processing time of operations were determined. Because their first generated set has unit processing times ($p_o = 1$ for each operation), we do not add this special case to the base sample. The second set, called "str-u99" in this paper,

used processing times that were drawn equally from the interval [1, 99], whereas the set "str-u200" has uniformly distributed processing times from the interval [1, 200]. Instances from the instance set "str-b" have processing times that were drawn from a binomial distribution with $n = 98$ and $p = 0.5$, resulting in processing times with an approximately normal distribution with the mean of 50. The last set of instances "str-nb" was generated by drawing the processing times from a negative binomial distribution with $r = 1$ and $p = 0.5$, shifted up by 1 (the processing times resulting from this distribution are approximately exponentially distributed). For all the instances, the machine sequence per job is generated by shuffling the array of machines. For each of the processing time distribution settings, five instances for each combination of n and m were created, leading to 845 instances per set. As instances with $n = 1$ or $m = 1$ lead to trivial or single machine instances, respectively, we remove these from the sets and only use the remaining 720 instances per set to create the second base sample E^{OR+STR} combining E^{OR} with the four described sets "str-u99", "str-u200", "str-b", and "str-nb".

Note that we do not consider the instances generated by Vela et al. (2021), as the instances generated by them are very small (most of them have only three jobs and four machines). In addition, the authors used a proprietary, untraceable file format.

4.2.2 Feature engineering

In the literature, a large number of features exist to describe JSSP instances (Table 6 in Appendix A1 provides an overview based on the features used in Mirshekarian & Šormaz, 2016 and Strassl & Musliu, 2022). However, when analyzing the existing CIGs, we found that the control attributes are the number of jobs, the number of machines, and the "type" of processing times. Therefore, we conclude that the number of jobs, the number of machines, and the central tendency and variability of the processing times are the most important and thus define features for reverse engineering that correspond to them. Note that the other features listed in Table 6 are of course helpful for example in solving an ASP for the JSSP (cf., e.g., Strassl & Musliu, 2022). Because Strassl & Musliu (2022) varied not only the interval from which the processing times were drawn, but also the type of distribution, we investigate one feature set that includes the mean and the coefficient of variation and another feature set that includes the mean and variance to estimate the alpha and beta parameters to define a beta distribution accordingly. A beta distribution is used because it is very flexible to express other distributions. The feature sets used for the reverse instance generation are listed in Table 4. For the SA, we use the feature set F^{SA} = {n, m, nmr, pM, pS, pCV, pVar, pSkew, pR, pGini}.

Table 4: Feature sets for the reverse engineering (JSSP)

Feature set	Features	Total explained variance (by PC1, by PC2)
$F^{n-m-pMS}$	*n, m, pM* (mean), *pS* (standard deviation)	0.73 (0.47, 0.26)
$F^{n-m-pMR}$	*n, m, pM, pR* (range)	0.75 (0.51, 0.25)
$F^{n-m-pMVar}$	*n, m, pM, pVar* (variance)	0.72 (0.44, 0.28)
$F^{n-m-pMCV}$	*n, m, pM, pCV* (coefficient of variation)	0.56 (0.28, 0.27)
$F^{n-nmr-pMVar}$	*n, nmr* (job to machine ratio), *pM, pVar*	0.90 (0.55, 0.35)

4.2.3 Dimension reduction and target point determination

The five feature sets in Table 4 are used to perform ISAs and the explained variances obtained are given in the last column. As can be seen, the feature set $F^{n-m-pMCV}$ has the lowest explained variance resulting from the dimension reduction and is therefore omitted from further investigation. Figure 12 shows the projection coefficients for the remaining four feature sets and sample E^{OR+STR}.

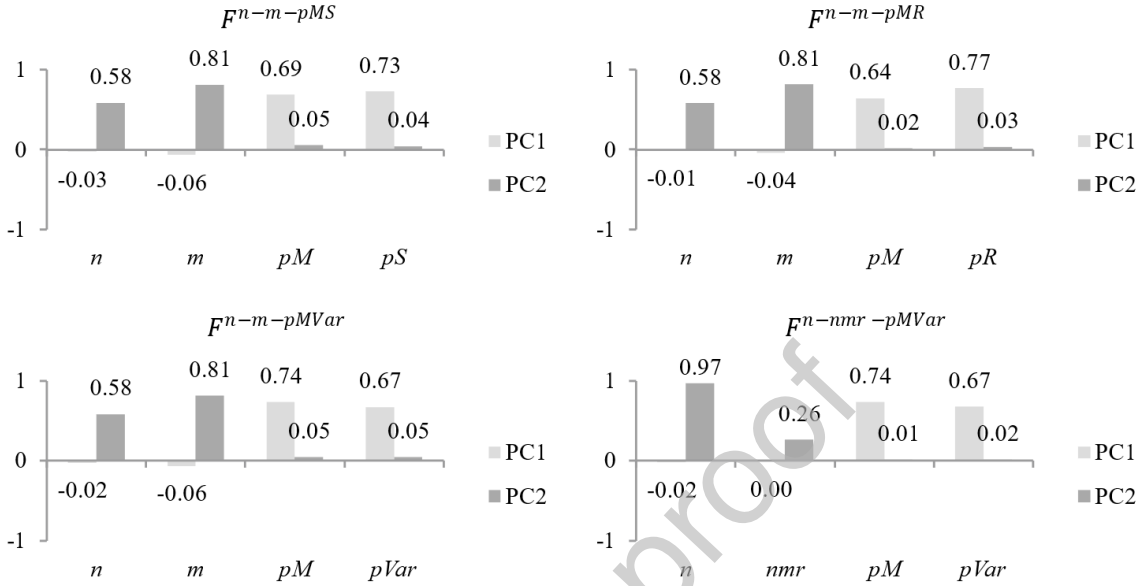

Figure 12: Projection coefficients for different feature sets (with identical sample set E^{OR+STR}).

As can be seen in all four graphs, one component of the PCA represents the information about the number of jobs and the number of machines, whereas the other component represents processing time information. It can also be seen that all four feature sets lead to a meaningful representation of the features required for reverse engineering and thus, all of them could be used for the reverse engineering.

As before, the target points T^2 for reverse engineering are determined with the parameters δ=0.1 and π=32, leading to a maximum number of 1,024 new instances (per feature set).

4.2.4 Reverse instance generation

Depending on the feature sets, different information is available for determining instance generation parameters. To obtain reasonable JSSP instances, we propose using the following bounds $LB^n = 10$, $LB^m = 5$, and $LB^p = 1$ for discarding (or adjusting).

For all four feature sets considered, the number of jobs is determined by $n_t = [\hat{n}_t]$. The number of machines is determined by $m_t = [\hat{m}_t]$, except for feature set $F^{n-nmr-pMVar}$, where $m_t = [\hat{n}_t / nmr_t]$. In the case of discarding instances, we only accept instances with feature values $n_t \geq LB^n$ and $m_t \geq LB^m$, and in the case of adjusting, maximum functions are used (cf. Section 4.1.4).

For $F^{n-m-pMS}$, the processing times of all operations are equally drawn from the interval restricted by the lower bound $lb_t^p = \left[pM_t - (\sqrt{12} \cdot pS_t) \cdot 0.5\right]$ and the upper bound $ub_t^p = \max\{lb_t^p + 1, \left[pM_t + (\sqrt{12} \cdot pS_t) \cdot 0.5\right]\}$. For $F^{n-m-pMR}$, the processing times are equally drawn from the interval bounded by $lb_t^p = [\, pM_t - pR_t \cdot 0.5]$ and $ub_t^p = \max\{lb_t^p + 1, [pM_t + pR_t \cdot 0.5]\}$. For these cases, when discarding unreasonable instances, we only accept instances with $lb_t^p \geq LB^p$. For

$F^{n-m-pMVar}$ and $F^{n-nmr-pMVar}$, we want to consider distributions for the processing times. Therefore, we use the reverse-engineered feature values pM_t and $pVar_t$ to estimate the parameters α_t and β_t of the corresponding beta distribution. Since beta distributions are defined between 0 and 1, we must first "unscale" these values. To do so, we define a "range scale" $rs = ub_t^p - lb_t^p$ by the minimum processing time $lb_t^p = LB^p$ and the (adjusted) maximum known operation processing (representing the upper bound) $ub_t^p = \max(p|\forall jobs \in E^{OR+STR}) \cdot 1.1$. With rs, we get the unscaled values for the mean $pM_t^{us} = (pM_t - lb_t^p)/rs$ and for the variance $pVar_t^{us} = pVar_t/rs^2$. With $s = (pM_t^{us} \cdot (1 - pM_t^{us}))/pVar_t^{us} - 1$, we can calculate the parameters $\alpha_t = pM_t^{us} \cdot s$ and $\beta_t = (1 - pM_t^{us}) \cdot s$. Whenever this leads to parameter values below zero (which are not appropriate for beta distributions) we set them to one. The unscaled processing times are then drawn by $p_o^{us} \sim Beta(\alpha_t, \beta_t)$ and then scaled to $p_o = \left[lb_t^p + p_o^{us} \cdot ub_t^p\right]$. Here, we discard instances if any of the drawn operation processing time p_o is smaller than LB^p.

The sequence of operations, i.e., the machine on which an operation must be executed, is determined by drawing a random permutation sequence of length m_t.

4.2.5 Validation

The first validation step is to analyze whether the newly created instances are located near the target points. To do this, we project the new instances onto the same instance space as used for the original ISA.

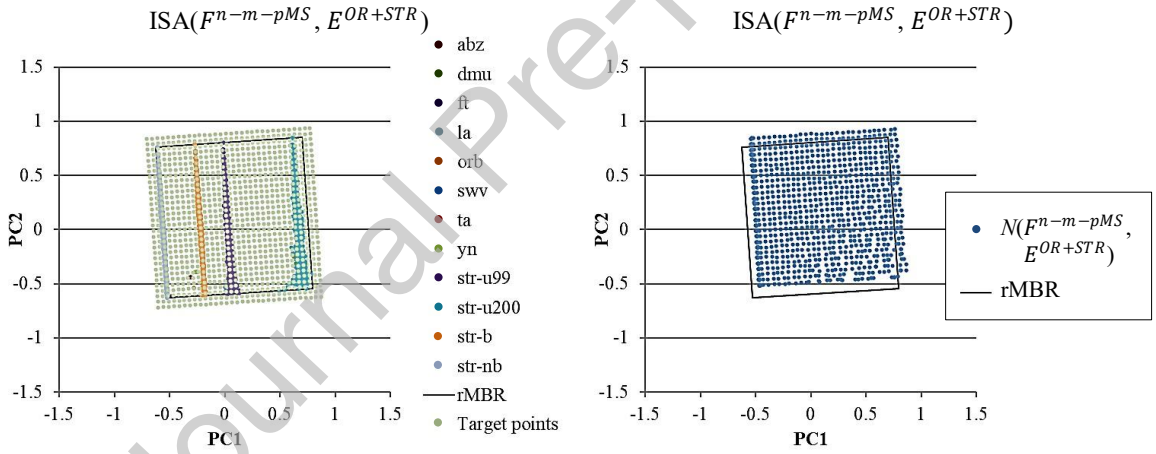

Figure 13: Base sample and target points T^2 (with δ=0.1 and π=32) (left side) and new instances (right side) both projected with feature set $F^{n-m-pMS}$.

Figure 13 and Figure 14 show that this is the case for the feature sets $F^{n-m-pMS}$ and $F^{n-nmr-pMVar}$ (it is also the case for $F^{n-m-pMR}$ and $F^{n-m-pMVar}$, but since their graphs are very similar to the ones presented here, the graphs are omitted). Note that in the following, we only consider sets where unreasonable instances are discarded: $|N(F^{n-m-pMS}, E^{OR+STR}|$=895 and $|N(F^{n-nmr-pMVar}, E^{OR+STR}|$=928.

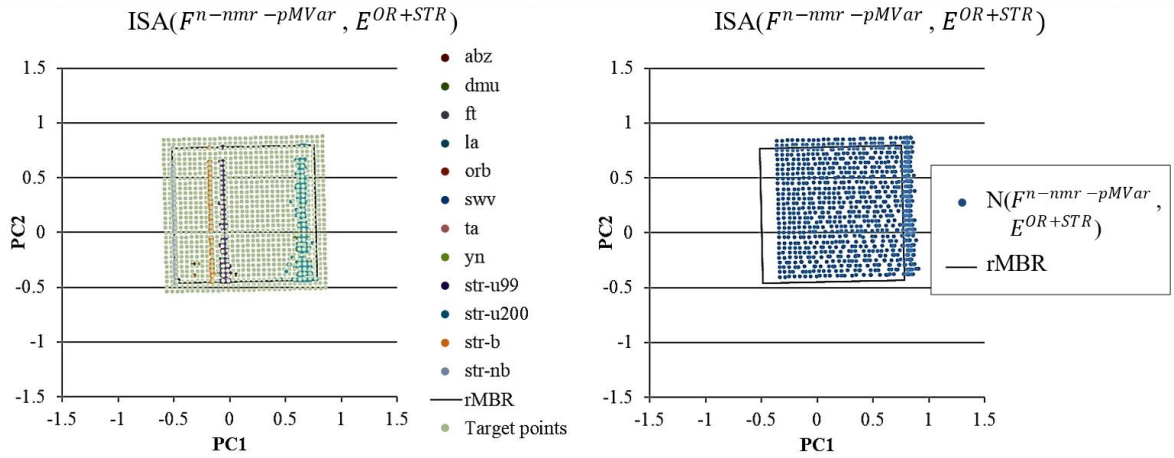

Figure 14: Base sample and target points T^2 (with δ=0.1 and π=32) (left side) and new instances (right side) both projected with feature set $F^{n-nmr-pMVar}$.

The second validation step is to repeat the ISA with the instance set $E^B \cup N$ and feature set F^B (used for the instance generation) to analyze whether the projection results in a "rectangular" shape. Figure 15 shows the ISAs for the feature sets $F^{n-m-pMS}$ and $F^{n-nmr-pMVar}$.

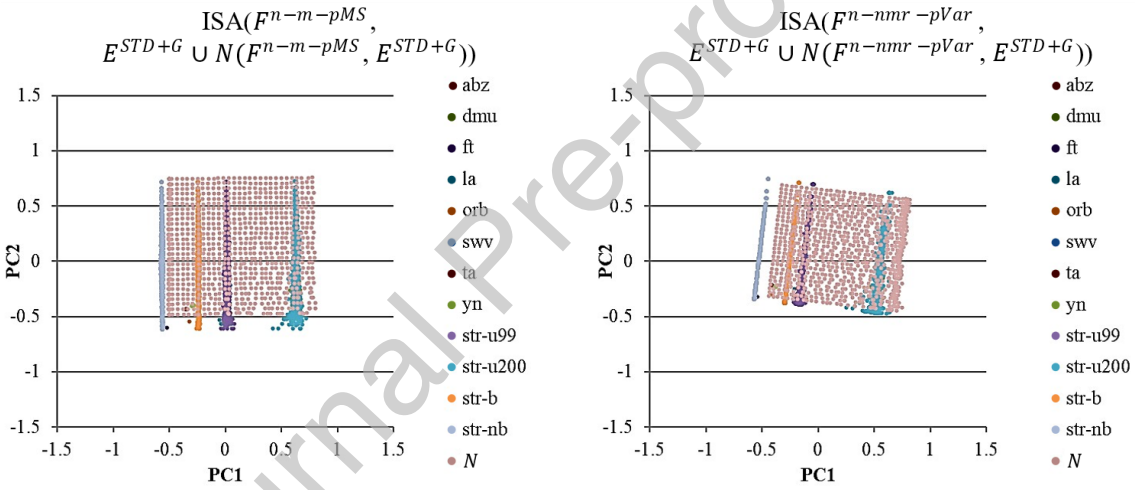

Figure 15: ISAs with the feature sets $F^{n-m-pMS}$ (left side) and $F^{n-nmr-pMVar}$ (right side) and the newly generated samples ($E^B \cup N$).

As can be seen in Figure 15, both graphs show the desired rectangular shape. The graphs also show that the instance sets "str-u99", "str-u200", "str-b", and "str-nb" contain instances with characteristics that are not reflected by the new instance sets (see the "blank" areas at the bottom and on the left). The special "position" of instance set "str-nb" at the left can be explained by their generally very small processing times (*pM* = 2.0). The other "blank" areas are explained by the lower bound of minimum 10 jobs per instance.

The third step of the validation is the comparison of the SAs with the base sample and the newly created sets: Figure 16 shows the violin plots for E^{OR+STR} and for the instance sets created by $F^{n-m-pMS}$ and $F^{n-nmr-pMVar}$, respectively. As can be seen from the SA for $N(F^{n-nmr-pMVar}, E^{OR+STR})$ in Figure 16, the diversity of the number of machines (*m*) is not sufficient for this feature set and thus, the reverse engineering by $F^{n-nmr-pMVar}$ is not appropriate. This observation also shows that the third step of the validation and the iterative and comprehensive observation of the instance generation process is essential.

However, the instance set $N(F^{n-m-pMS}, E^{OR+STR})$ provides the desired high diversity of feasible and reasonable instances (see Figure 16), and therefore constitutes the new instance set generated by RIG.

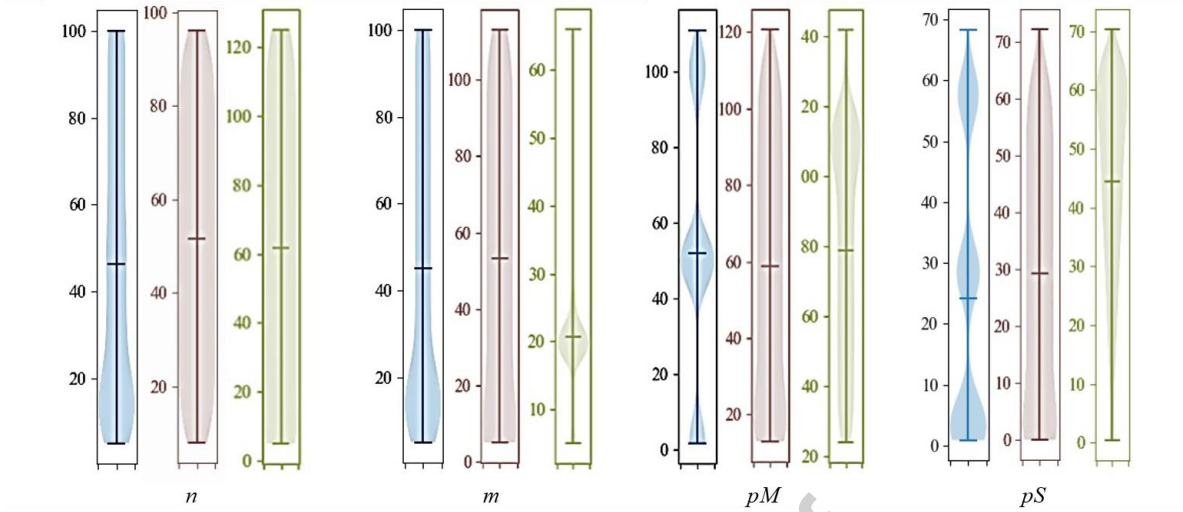

Figure 16: Violin plots of selected features for the base sample E^{OR+STR} (blue) and for the new instance sets $N(F^{n-m-pMS}, E^{OR+STR})$ (red) and $N(F^{n-nmr-pMVar}, E^{OR+STR})$ (green).

4.3 Parallel serial-batch scheduling problem with incompatible job families

The parallel serial-batch scheduling problem with incompatible job families (PSBIJF) is a relatively new problem first described in Gahm et al. (2021). The PSBIJF problem considers identical parallel machines with the serial-batch processing characteristic (i.e., the processing time of a batch is equal to the sum of the processing times of all jobs grouped in a batch), limited batch capacities, batch availability, and sequence-dependent setup times combined with incompatible job families (jobs from different families cannot be processed in the same batch) and arbitrary job sizes (batch capacity requirements). The primary objective is to minimize total weighted tardiness. For a comprehensive overview of such problems, see Wahl et al. (2024).

4.3.1 Base samples

The considered base sample E^{UGWT} consists of 93,360 instances proposed by Uzunoglu et al. (2023) and available for download at Mendeley data (Gahm, 2022). The instance set consists of three subsets: UGWT-S has 22,320 instances with the number of jobs $n \in \{15, 30, 60\}$, the number of machines $m \in \{1, 3, 4, 5\}$, and the number of job families $q \in \{3, 5, 10\}$, the set UGWT-L contains 57,600 instances with $n \in \{100, 200, 400\}$, $m \in \{1, 3, 4, 5, 10\}$and $q \in \{3, 5, 10, 20\}$, and the set UGWT-XL contains 13,440 instances with $n \in \{800, 1{,}600, 3{,}200\}$, $m \in \{5, 10, 20\}$, and $q \in \{10, 20, 40\}$.

4.3.2 Feature engineering

The seven main attributes (characteristics) of the instance generation procedure used by Uzunoglu et al. (2023) are the number of jobs, the number of machines, the number of job families, the tendency and variability of batch capacity requirements, the setup time severity, the tendency and variability of the due dates, and the assignment of jobs to families. The complete set of features available to describe PSBIJF instances is given in Table 7 in Appendix A2 (see also Uzunoglu et al., 2023). Since the possible combinations of features required for the reverse engineering are very large, we present only a subset of the

investigated feature sets (with the best overall results) in Table 5. For the SA, we use all features listed in Table 7.

Table 5: Feature sets for the reverse engineering (PSBIJF)

Feature set	Features	Total explained variance (by PC1, by PC2)
$F^{aRN-sMS-dMS-jfMVar}$	*n, m, q, rnS, rnM, rnL, rnXL, sM, sS, dM, dS, jfM, jfVar*	0.60 (0.38, 0.22)
$F^{aRN-sMVar-dMVar-jfMVar}$	*n, m, q, rnS, rnM, rnL, rnXL, sM, sVar, dM, dVar, jfM, jfVar*	0.61 (0.38, 0.23)
$F^{aMVar-sMS-dMS-jfMVar}$	*n, m, q, aM, aVar, sM, sS, dM, dS, jfM, jfVar*	0.56 (0.31, 0.27)
$F^{aMVar-sMVar-dMvar-jfMVar}$	*n, m, q, aM, aVar, sM, sVar, dM, dVar, jfM, jfVar*	0.58 (0.30, 0.28)

4.3.3 Dimension reduction and target point determination

The four feature sets in Table 5 are used to perform ISAs and the explained variances obtained are given in the last column. As the graphs of the projection coefficients do not provide any meaningful insight into the differences between the four feature sets, we do not present them here.

The target points T^2 for reverse engineering are determined with the parameters δ=0.1 and π=32 (see Figure 5). This results in a maximum number of 1,024 new instances.

4.3.4 Reverse instance generation

Depending on the feature set, different information is available to determine the required instance generation parameters. To obtain reasonable PSBIJF instances, we propose using the bounds $LB^n = 15$, $LB^m = 1$, and $LB^q = 3$ for discarding instances. Regarding the other instance parameters, we generally use a lower bound equal to one.

For all feature sets, the number of jobs for a new instance related to target point *t* is determined by $n_t = [\hat{n}_t]$, the number of machines is determined by $m_t = [\hat{m}_t]$, and the number of job families is determined by $q_t = [\hat{q}_t]$.

For the feature sets indicated by "*aRN*", the batch capacity requirement is determined by the reverse-engineered relative proportions of jobs of size S, M, L, and XL, respectively. Together with n_t, we can determine the corresponding number of jobs (e.g., $\widehat{nS}_t = [\mathrm{rnS}_t \cdot n_t]$) and an adjustment mechanism guarantees that $\hat{n}S_t + \hat{n}M_t + \hat{n}L_t + \hat{n}XL_t = \hat{n}_t$. Using these numbers, the batch capacity requirements are drawn from discrete uniform distributions bounded by 1 and 12 for S, by 1 and 25 for M, by 1 and 50 for L, and by 13 and 38 for XL jobs. For feature sets indicated by "$aMVar$", batch capacity requirements are determined by aM_t and $aVar_t$ and the procedure described for processing times in Section 4.2.4. This procedure is also used for setup times, due dates, and job to family assignments if appropriate (denoted by "$...MVar$" in feature set names). "$...MS$" in a feature set name indicates that the values are drawn from uniform distributions derived by means and standard deviations (see Section 4.1.4). To restrict the new instance set to reasonable instances, we discard instances with $n_t < LB^n$, $m_t < LB^m$, $q_t < LB^q$ and any instance with any parameter value smaller than one.

4.3.5 Validation

Although the feature sets indicated by "*aRN*" provide a slightly higher explained variance, the feature sets indicated by "$aMVar$" showed to be the better overall results and therefore only $F^{aMVar-sMVar-dMVar-jfMVar}$ is validated here. The projection of the base sample (and target points) and the new instances onto the same instance space shows that the new instances are not perfectly located at the target points (see Figure 17). However, they cover the desired area quite well.

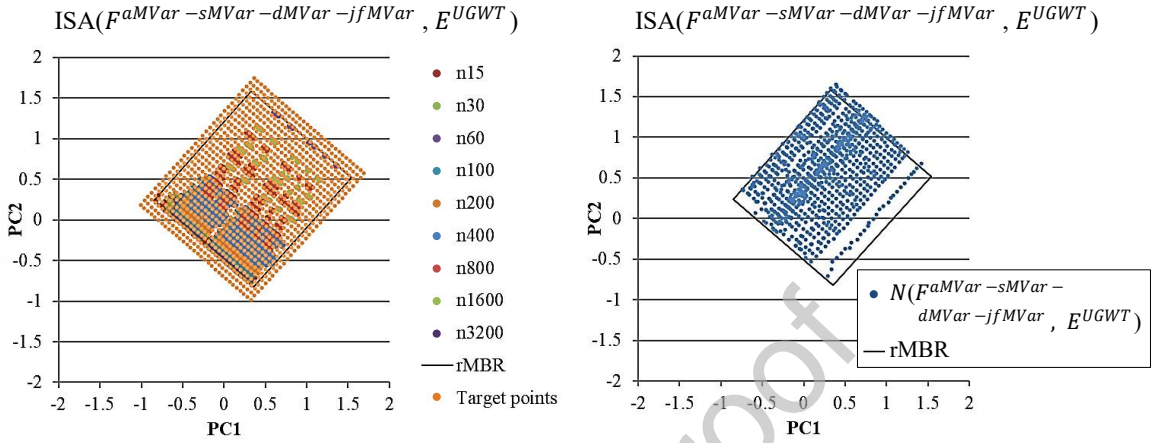

Figure 17: Base sample and target points T^2 (with δ=0.1 and π=32) (left side) and new instances (right side) projected with feature set $F^{aMVar-sMVar-dMVar-jfMVar}$.

In Figure 18, we show two ISAs with feature set $F^{aMVar-sMVar-dMVar-jfMVar}$ and $(E^B \cup N)$: the base sample on the left side and the new instances on the right side. We can see that the new instances fill gaps in some regions of the instance space (top right). However, we can also see that other regions are already very well covered by the original instances. This is due to the large number of instances in instance set UGWT-L (with 57,600 instances with $n \in \{100, 200, 400\}$) and the diversity that this brings. The advantage of our approach here is that much fewer instances (883; with discarding unreasonable instances) are required to cover the instance space, and thus the effort to evaluate solution methods is much lower.

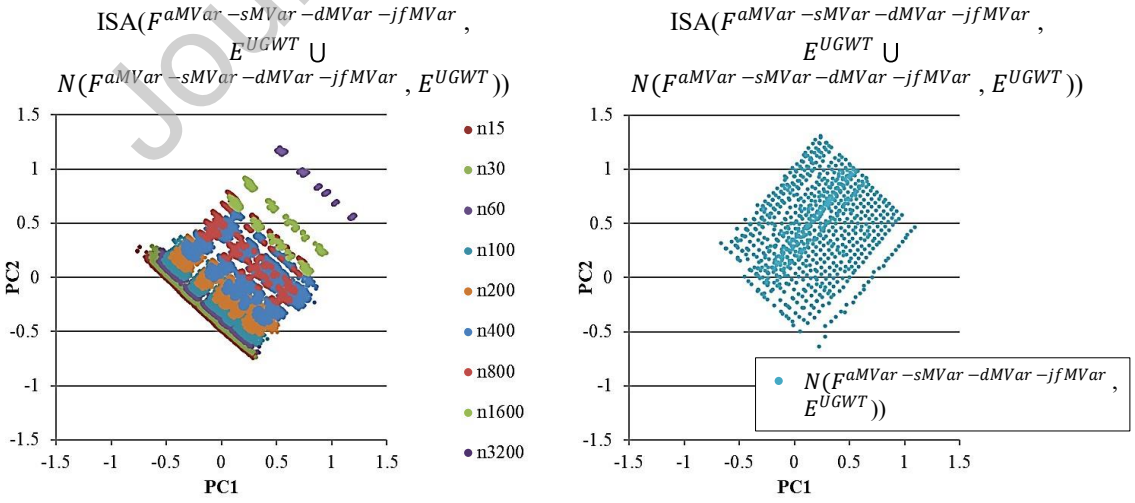

Figure 18: Comparison of ISAs with the original base sample and newly generated instance sample (projected with the feature set $F^{aMVar-sMVar-dMVar-jfMVar}$).

The desired diversity of the newly generated instances is shown by the violin plots in Figure 19.

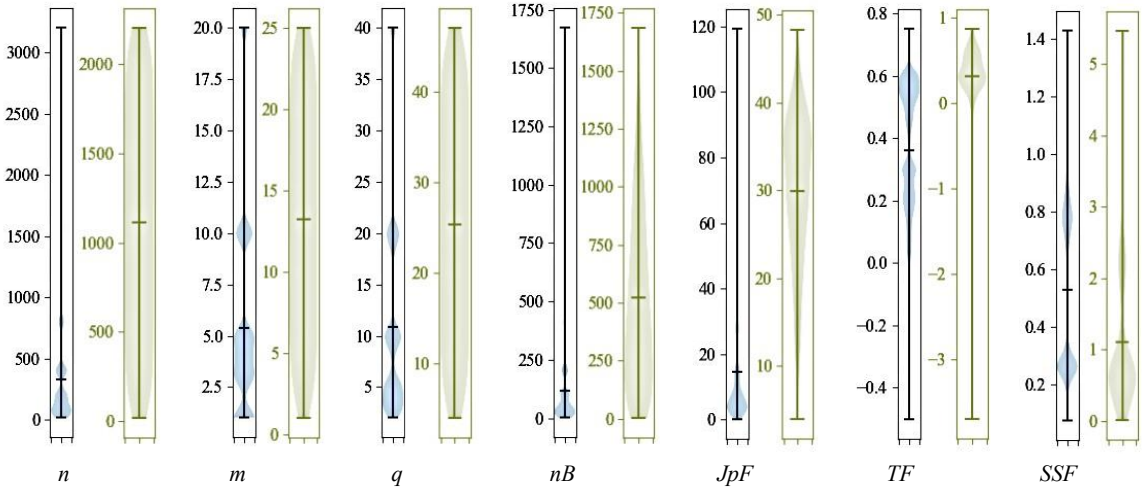

Figure 19: Violin plots of selected features for the base sample E^{UGWT} (blue) and for the new instance set N($F^{aMVar-sMVar-dMVar-jfMVar}$, E^{UGWT}) (red).

Please note that the maximum number of jobs for the new instances is much lower than for the original instances. This is because the majority of the base sample has fewer than 2,000 jobs. To compensate for this, an adaptation mechanism could be used to "artificially" increase the number of jobs. However, as can be seen from the other selected features, the tendency and variability of all the three main instance characteristics (*n*, *m*, and *q*) are remarkably more diverse in the new instance set. Furthermore, the greater diversity of the *nb*, *JpF*, and *TF* features indicates that the new PSBIJF instances are more challenging, as it is more likely that more batches will have to be scheduled and that due dates will be tighter (lower *TF* values indicate tighter due dates). Moreover, the generally higher setup time severity (*SSF*), including some extreme values, suggests additional challenges in solving the new instances. Accordingly, the new instance set generated by RIG for the PSBIJF, is the set $N(F^{aMVar-sMVar-dMVar-jfMVAR}, E^{UGWT})$.

5 Conclusions and further research directions

This paper presents the RIG method, a new approach for generating diverse, feasible, and reasonable machine scheduling instances using reverse engineering from instance space analysis. By using ISA to identify gaps or underrepresented areas within the instance space, RIG systematically constructs new instances with desired properties, overcoming the limitations of some existing instance generation methods to provide reasonable instances, i.e., by bounding and/or adjusting individual or combinations of instance generation procedure parameters. The proposed method has been validated on three scheduling problems: SMTWTP, JSSP, and PSBIJF. The results showed that RIG effectively fills gaps in the instance space and expands its boundaries, providing new comprehensive sets of instances. Additionally, we demonstrated that the new instances of the SMTWTP are useful and challenging for fulfilling the various purposes of solution method performance testing. Unlike traditional methods, RIG ensures unbiased generation without favoring specific solution methods, which enhances its applicability to diverse optimization scenarios.

Future research should explore extending RIG to other problem domains and further refining the feature engineering to maximize generalizability. In this context, the influence of the type of features (i.e., continuous, integer or binary) on the applicability of the RIG method needs to be further investigated.

Furthermore, the integration of automated feedback loops for iterative improvement and additional mechanisms for improving the reverse engineering of generation parameters may provide further insights and benefits. For example, the merging of our approach with the optimization-based PILOT method proposed by Muñoz et al. (2018) to provide improved projection coefficients. Because our RIG method preserves the general characteristics of a sample set with fewer samples while avoiding "clusters" of samples in certain regions of the instance space, it could be used to prevent sampling bias in machine learning applications (which is especially true when instances are discarded). The effects of reduced sampling bias on solution method performance testing and particularly machine learning is an interesting topic to investigate in detail in the future.

Overall, RIG represents a meaningful advancement in the generation of problem instances and promotes rigorous and comprehensive benchmarking in machine scheduling.

References

Alipour, H., Muñoz, M. A., & Smith-Miles, K. (2023). Enhanced instance space analysis for the maximum flow problem. *Eur. J. Oper. Res., 304*(*2), 411–428.* doi:10.1016/j.ejor.2022.04.012.

Beasley, J. E. (1990). OR-Library: Distributing Test Problems by Electronic Mail. *J. Oper. Res. Soc., 41*(*11), 1069.* doi:10.2307/2582903.

Bierwirth, C., Mattfeld, D. C., & Watson, J.-P. (2004). Landscape Regularity and Random Walks for the Job-Shop Scheduling Problem. (21–30) In T. Kanade, J. Kittler, J. M. Kleinberg, F. Mattern, J. C. Mitchell, O. Nierstrasz, … (Eds.), *Lecture Notes in Computer Science. Evolutionary Computation in Combinatorial Optimization*, Berlin, Heidelberg: Springer.

Bowly, S., Smith-Miles, K., Baatar, D., & Mittelmann, H. (2020). Generation techniques for linear programming instances with controllable properties. *Math. Prog. Comp., 12*(*3), 389–415.* doi:10.1007/s12532-019-00170-6.

Coster, A. de, Musliu, N., Schaerf, A., Schoisswohl, J., & Smith-Miles, K. (2022). Algorithm selection and instance space analysis for curriculum-based course timetabling. *J. Sched., 25*(*1), 35–58.* doi:10.1007/s10951-021-00701-x.

Cotta, C., & Moscato, P. (2003). A mixed evolutionary-statistical analysis of an algorithm's complexity. *Appl. Math. Lett., 16*(*1), 41–47.* doi:10.1016/S0893-9659(02)00142-8.

Gahm, C. (2022). Extended instance sets for the parallel serial-batch scheduling problem with incompatible job families, sequence-dependent setup times, and arbitrary sizes. *Mendeley Data*, (*1).* doi:10.17632/rxc695hj2k.1.

Gahm, C. (2025). Base samples and new instances generated by reverse engineering from instance space analysis. *Mendeley Data, V2.* doi:10.17632/j7g23t4ynb.2.

Gahm, C., Kanet, J. J., & Tuma, A. (2019). On the flexibility of a decision theory-based heuristic for single machine scheduling. *Comput. Oper. Res., 101, 103–115.* doi:10.1016/j.cor.2018.09.005.

Gahm, C., Wahl, S., & Tuma, A. (2021). Scheduling parallel serial-batch processing machines with incompatible job families, sequence-dependent setup times and arbitrary sizes. *Int. J. Prod. Res., 60*(*17), 5131–5154.* doi:10.1080/00207543.2021.1951446.

Gao, W., Nallaperuma, S., & Neumann, F. (2021). Feature-Based Diversity Optimization for Problem Instance Classification. *Evol. Comput., 29, 107–128.* doi:10.1162/evco_a_00274.

Geiger, M. J. (2010). *New Instances for the Single Machine Total Weighted Tardiness Problem: Research Report RR-10-03-01*, Helmut Schmidt University. Hamburg.

Graham, R. L., Lawler, E. L., Lenstra, J. K., & Kan, A. (1979). Optimization and Approximation in Deterministic Sequencing and Scheduling: a Survey. (287–326) In *Annals of Discrete Mathematics. Discrete Optimization II*, Elsevier.

Hall, N. G., & Posner, M. E. (2001). Generating Experimental Data for Computational Testing with Machine Scheduling Applications. *Oper. Res., 49*(*6), 854–865.* doi:10.1287/opre.49.6.854.10014.

Hall, N. G., & Posner, M. E. (2010). The Generation of Experimental Data for Computational Testing in Optimization. (73–101) In T. Bartz-Beielstein, M. Chiarandini, L. Paquete, & M. Preuss (Eds.), *Experimental Methods for the Analysis of Optimization Algorithms*, Berlin, Heidelberg: Springer.

Hooker, J. N. (1994). Needed: An Empirical Science of Algorithms. *Oper. Res.*, *42*(*2*), *201–212.* doi:10.1287/opre.42.2.201.

Hooker, J. N. (1995). Testing heuristics: We have it all wrong. *J Heuristics*, *1*(*1*), *33–42.* doi:10.1007/BF02430364.

Jooken, J., Leyman, P., & Causmaecker, P. de. (2022). A new class of hard problem instances for the 0–1 knapsack problem. *Eur. J. Oper. Res.*, *301*(*3*), *841–854.* doi:10.1016/j.ejor.2021.12.009.

Kang, Y., Hyndman, R. J., & Smith-Miles, K. (2017). Visualising forecasting algorithm performance using time series instance spaces. *International Journal of Forecasting*, *33*(*2*), *345–358.* doi: 10.1016/j.ijforecast.2016.09.004.

Kletzander, L., Musliu, N., & Smith-Miles, K. (2021). Instance space analysis for a personnel scheduling problem. *Ann Math Artif Intell*, *89*(*7*), *617–637.* doi:10.1007/s10472-020-09695-2.

Kolisch, R., Schwindt, C., & Sprecher, A. (1999). Benchmark Instances for Project Scheduling Problems. (197–212) In F. S. Hillier & J. Węglarz (Eds.), *International Series in Operations Research & Management Science. Project Scheduling*, Boston, MA: Springer.

Lechien, T., Jooken, J., & Causmaecker, P. de. (2023). Evolving test instances of the Hamiltonian completion problem. *Comput. Oper. Res.*, *149*, *106019.* doi: 10.1016/j.cor.2022.106019.

Liu, C., Smith-Miles, K., Wauters, T., & Costa, A. M. (2023). Instance space analysis for 2D bin packing mathematical models. *Eur. J. Oper. Res.* doi: 10.1016/j.ejor.2023.12.008.

Lopes, L., & Smith-Miles, K. (2013). Generating Applicable Synthetic Instances for Branch Problems. *Oper. Res.*, *61*(*3*), *563–577.* doi:10.1287/opre.2013.1169.

Lou, Y., & Yuen, S. Y. (2019). On constructing alternative benchmark suite for evolutionary algorithms. *Swarm and Evolutionary Computation*, *44*, *287–292.* doi: 10.1016/j.swevo.2018.04.005.

Martello, S., Pisinger, D., & Toth, P. (1999). Dynamic Programming and Strong Bounds for the 0-1 Knapsack Problem. *Manage Sci*, *45*(*3*), *414–424.* doi:10.1287/mnsc.45.3.414.

Mersmann, O., Bischl, B., Trautmann, H., Wagner, M., Bossek, J., & Neumann, F. (2013). A novel feature-based approach to characterize algorithm performance for the traveling salesperson problem. *Ann Math Artif Intell*, *69*(*2*), *151–182.* doi:10.1007/s10472-013-9341-2.

Mirshekarian, S., & Šormaz, D. N. (2016). Correlation of job-shop scheduling problem features with scheduling efficiency. *Expert Syst. Appl.*, *62*, *131–147.* doi: 10.1016/j.eswa.2016.06.014.

Muñoz, M. A., & Smith-Miles, K. (2017). Performance Analysis of Continuous Black-Box Optimization Algorithms via Footprints in Instance Space. *Evol. Comput.*, *25*(*4*), *529–554.* doi:10.1162/EVCO_a_00194.

Muñoz, M. A., & Smith-Miles, K. (2020). Generating New Space-Filling Test Instances for Continuous Black-Box Optimization. *Evol. Comput.*, *28*(*3*), *379–404.* doi:10.1162/evco_a_00262.

Muñoz, M. A., Villanova, L., Baatar, D., & Smith-Miles, K. (2018). Instance spaces for machine learning classification. *Mach Learn*, *107*(*1*), *109–147.* doi:10.1007/s10994-017-5629-5.

Muñoz, M. A., Yan, T., Leal, M. R., Smith-Miles, K., Lorena, A. C., Pappa, G. L., & Rodrigues, R. M. (2021). An Instance Space Analysis of Regression Problems. *ACM Trans. Knowl. Discov. Data*, *15*(*2*), *1–25.* doi:10.1145/3436893.

Neuenfeldt Júnior, A., Silva, E., Gomes, A. M., Soares, C., & Oliveira, J. F. (2019). Data mining-based framework to assess solution quality for the rectangular 2D strip-packing problem. *Expert Syst. Appl.*, *118*, *365–380.* doi: 10.1016/j.eswa.2018.10.006.

Reilly, C. H. (2009). Synthetic Optimization Problem Generation: Show Us the Correlations! *INFORMS J. Comput.*, *21*(*3*), *458–467.* doi:10.1287/ijoc.1090.0330.

Rice, J. R. (1976). The Algorithm Selection Problem. *Adv. in Comp.*, *15*, *65–118.* doi:10.1016/S0065-2458(08)60520-3.

Scherer, M. E., Hill, R. R., Lunday, B. J., Cox, B. A., & White, E. D. (2024). Verifying new instances of the multidemand multidimensional knapsack problem with instance space analysis. *Comput. Oper. Res.*, *162*, *106477.* doi: 10.1016/j.cor.2023.106477.

Shand, C., Allmendinger, R., Handl, J., Webb, A., & Keane, J. (2022). HAWKS: Challenging Benchmark Sets for Cluster Analysis. *IEEE Trans. Evol. Computat., 26*(*6*), *1206–1220.* doi:10.1109/TEVC.2021.3137369.

Silva, E., Oliveira, J. F., & Wäscher, G. (2014). 2DCPackGen: A problem generator for two-dimensional rectangular cutting and packing problems. *Eur. J. Oper. Res.*, *237*(*3*), *846–856.* doi: 10.1016/j.ejor.2014.02.059.

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

Smith-Miles, K., & Bowly, S. (2015). Generating new test instances by evolving in instance space. *Comput. Oper. Res.*, *63*, *102–113.* doi: 10.1016/j.cor.2015.04.022.

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

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

Smith-Miles, K., & Muñoz, M. A. (2023). Instance Space Analysis for Algorithm Testing: Methodology and Software Tools. *ACM Comput. Surv.*, *55*(*12*), *1–31.* doi:10.1145/3572895.

Smith-Miles, K., & van Hemert, J. (2011). Discovering the suitability of optimisation algorithms by learning from evolved instances. *Ann Math Artif Intell*, *61*(*2*), *87–104.* doi:10.1007/s10472-011-9230-5.

Smith-Miles, K., van Hemert, J., & Lim, X. Y. (2010). Understanding TSP Difficulty by Learning from Evolved Instances. (266–280) In D. Hutchison, T. Kanade, J. Kittler, J. M. Kleinberg, F. Mattern, J. C. Mitchell, … (Eds.), *Lecture Notes in Computer Science. Learning and Intelligent Optimization*, Berlin, Heidelberg: Springer.

Strassl, S., & Musliu, N. (2022). Instance space analysis and algorithm selection for the job shop scheduling problem. *Comput. Oper. Res.*, *141*, *105661.* doi: 10.1016/j.cor.2021.105661.

Sun, Y., Esler, S., Thiruvady, D., Ernst, A. T., Li, X., & Morgan, K. (2024). Instance space analysis for the car sequencing problem. *Ann. Oper. Res.*, *341*(*1*), *41–69.* doi:10.1007/s10479-022-04860-8.

Uzunoglu, A., Gahm, C., Wahl, S., & Tuma, A. (2023). Learning-augmented heuristics for scheduling parallel serial-batch processing machines. *Comput. Oper. Res.*, *151*, *106122.* doi: 10.1016/j.cor.2022.106122.

van Bulck, D., & Goossens, D. (2023). The international timetabling competition on sports timetabling (ITC2021). *Eur. J. Oper. Res.*, *308*(*3*), *1249–1267.* doi: 10.1016/j.ejor.2022.11.046.

van Hemert, J. I. (2006). Evolving combinatorial problem instances that are difficult to solve. *Evol. Comput.*, *14*(*4*), *433–462.* doi:10.1162/evco.2006.14.4.433.

van Hoorn, J. J. (2018). The Current state of bounds on benchmark instances of the job-shop scheduling problem. *J. Sched.*, *21*(*1*), *127–128.* doi:10.1007/s10951-017-0547-8.

Vela, A., Cruz-Duarte, J. M., Ortiz-Bayliss, J. C., & Amaya, I. (2021). Tailoring Job Shop Scheduling Problem Instances Through Unified Particle Swarm Optimization. *IEEE Access*, *9*, *66891–66914.* doi:10.1109/ACCESS.2021.3076426.

Wahl, S., Gahm, C., & Tuma, A. (2024). Serial- and hierarchical-batch scheduling: a systematic review and future research directions. *Int. J. Prod. Res.*, *1–31.* doi:10.1080/00207543.2024.2432473.

Weise, T., Chen, Y., Li, X., & Wu, Z. (2020). Selecting a diverse set of benchmark instances from a tunable model problem for black-box discrete optimization algorithms. *Appl. Soft Comput.*, *92*, *106269.* doi: 10.1016/j.asoc.2020.106269.

Yap, E., Muñoz, M. A., & Smith-Miles, K. (2022). Informing Multiobjective Optimization Benchmark Construction Through Instance Space Analysis. *IEEE Trans. Evol. Computat.*, *26*(*6*), *1246–1260.* doi:10.1109/TEVC.2022.3205165.

Appendix

Appendix A1 JSSP feature space

Table 6: Feature space *F* for the JSSP

Definition	Description
n	Number of jobs (n)
m	Number of machines (m)
nmr	Job to machine ratio: NumJ / NumM
Lb-Ta	Lower bound on the makespan: $maxLb = \max\left\{\max_i(b_i + T_i + a_i), \max_j\left(\sum_{i=1}^{m} p_{i,j}\right)\right\}$, with $b_i = \min_j\left(pb_{j,i}\right)$, with $pb_{j,i}$ equal to the sum of processing times until job j is ready to be processed on machine I, with $T_i = \sum_{j=1}^{j\leq n} p_{i,j}$ (total processing time on machine *i*), and with $a_i = \min_j\left(pa_{j,i}\right)$, with $pa_{j,i}$ equal to the sum of processing times until job *j* is completed after being processed on machine *i*.
pM, pS, pVar, pCV, pMin, pMax, pR, pSkew, pGini	Statistics (mean, standard deviation, variance, coefficient of variation, minimum, maximum. range, skewness and Gini-coefficient) for the processing time per operation.
pmM, pmS, ...	Statistics for the processing time per machine.
pjMean, pjS, …	Statistics for the processing time per job (sum of operation processing times).
posM, posS, …	Statistics for the processing time per operation slot (e.g., the first operation slot contains the first operation of each job independent of the designated machine)
OSMM_M, OSMM_S, …	Statistics for the number of missing machines per operation slot normalized by the number of machines.
OSRM_M, OSRM_S, …	Statistics for the number of repeated machines per operation slot normalized w.r.t the number of machines.
OSRMA_M, OSRMA_, …	Statistics for the number of repeated machines per operation slot (amplified) normalized w.r.t the number of machines.
OSRMAp_M, OSRMAp_S, …	Statistics for the number of repeated machines per operation slot (amplified) multiplied by the mean of the corresponding operation processing time and normalized by the number of machines.
MLDU_M, MLDU_S, …	Statistics for the machine load uniformity normalized by the number of machines.
MLDV_M, MLDV_S, …	Statistics for the machine load voids normalized by the number of machines.
MLDVA_M, MLDVA_S, …	Statistics for the machine load voids (amplified) normalized by the number of machines

VerDegree_M, VerDegree_S, …	Statistics for the vertex degree of the unidirect disjunctive graph (Number of edges in the disjunctive graph that are incident to the regarded vertex v)
BC_M, BC_S, …	Statistics for the betweenness-centrality of the unidirect disjunctive graph (Relation of how often a vertex v appears on the shortest paths between two other vertices to the number of shortest paths)
Graph-Density	Density of the unidirect disjunctive graph ($dens(G) = \frac{2\|E\|}{\|V\|(\|V\|-1)}$)
RMSD	Root-mean-squared deviation of the operations slots to machine number (squared deviation of a machine sequence from the standard machine sequence 1, 2, 3, …; normalized by the number of operations)
pM_Ms_S	Standard deviation of the means of the processing time per machine: If the value is high, the processing times are very different, if the value is low, the processing times are similar
pM_Ss_M	Mean of the standard deviations of the processing time per machine: If the value is high, the processing times have a high range, if the value is low, the processing times have a low range.
pOS_Ms_S	Standard deviation of the means of the processing time per operation slot: If the value is high, the processing times are very different. If the value is low, the processing times are similar.
pOS_Ss_M	Mean of the standard deviations of the processing time per operation slot: If the value is high, the processing times have a high range. If the value is low, the processing times have a low range.
Increase-Factor	+1 added if the following machine in the machine number has a higher machine number than the current; normalized w.r.t. the number of jobs and number of machines
Slope-Factor	Machine numbers are multiplied with the number of the operation slot and added together, normalized w.r.t the number of jobs and number of machines

Appendix A2 PSBIJF feature space

Table 7: Feature space *F* for the PSBIJF

Definition	Description
n	Number of jobs
m	Number of machines
q	Number of incompatible job families
bc	Maximum batch capacity
appM	Approximated makespan
TF	Tardiness factor: 1 - (*dM* / *appM*)
RDD	Due date range: *dMax* – *dMin*
JpB	Approximated number of jobs per batch: *bc* / *aM*
nB	Approximated number of batches: *n*/*JpB*
BoJ	Batch occupation per job: BoJ = *aM* / *bc*
SSF	Setup time severity factor: *sM* / (*pM* * *JpB*)
SDF	Setup time diversity factor: *sR* / *sM*
pM, pS, pVar, pCV, pMin, pMax, pR, pSkew, pGini	Statistics (mean, standard deviation, variance, coefficient of variation, minimum, maximum. range, skewness and Gini-coefficient) for the processing time per job.
dM, dS, …	Statistics of job due dates
aM, aS, …	Statistics of batch capacity requirements (area) per job (*aJ*)
sM, sS, …	Statistics of setup times
jfM, jfS, …	Statistics of jobs per family
rnS	Relative number of jobs with *aJ* <= 0.1 bc
rnM	Relative number of jobs with 0.1 bc < *aJ* <= 0.33 bc
rnL	Relative number of jobs with 0.33 bc < *aJ* <= 0.5 bc
rnXL	Relative number of jobs with 0.5 bc < *aJ*

Appendix A3 SMTWTP feature projection

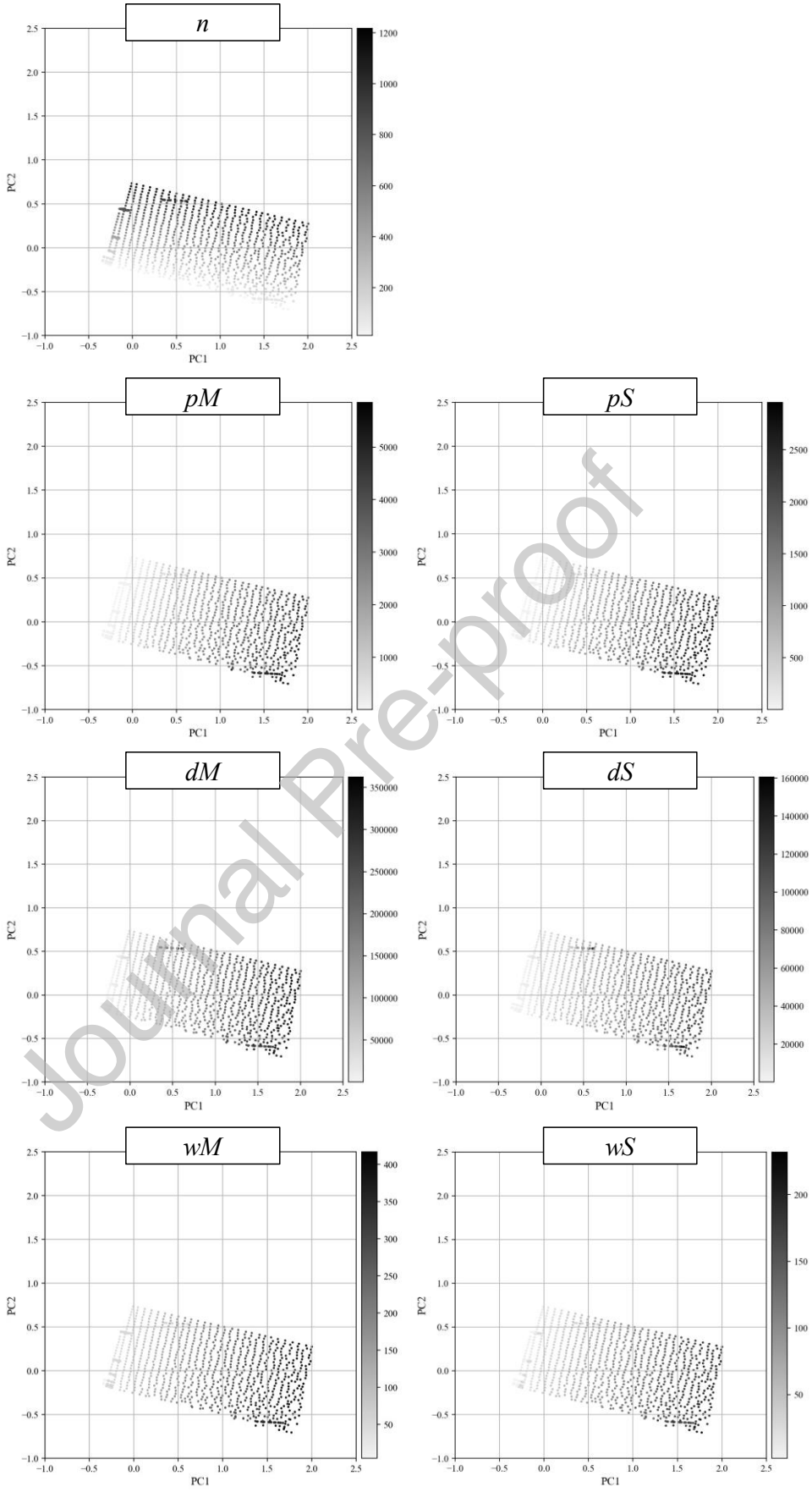

Figure 20: Feature projection for the combined instance set $E \cup N$

Declarations of interest: none