Author/s:
Kletzander, L;Musliu, N;Smith-Miles, K

Title:
Instance space analysis for a personnel scheduling problem

Date:
2020-04-24

Citation:
Kletzander, L., Musliu, N. & Smith-Miles, K. (2020). Instance space analysis for a personnel scheduling problem. ANNALS OF MATHEMATICS AND ARTIFICIAL INTELLIGENCE, 89 (7), pp.617-637. https://doi.org/10.1007/s10472-020-09695-2.

Persistent Link:
https://hdl.handle.net/11343/252054

Annals of Mathematics and Artificial Intelligence
https://doi.org/10.1007/s10472-020-09695-2

Instance space analysis for a personnel scheduling problem

Lucas Kletzander[1] · **Nysret Musliu**[1] · **Kate Smith-Miles**[2]

Published online: 24 April 2020

Abstract
This paper considers the Rotating Workforce Scheduling Problem, and shows how the strengths and weaknesses of various solution methods can be understood by the in-depth evaluation offered by a recently developed methodology known as Instance Space Analysis. We first present a set of features aiming to describe hardness of test instances. We create a new, more diverse set of instances based on an initial instance space analysis that reveals gaps in the instance space, and offers the opportunity to generate additional instances to add diversity to the test suite. The results of three algorithms on our extended instance set reveal insights based on this visual methodology. We observe different regions of strength and weakness in the instance space for each algorithm, as well as a phase transition from feasible to infeasible instances with more challenging instances at the phase transition boundary. This rigorous and insightful approach to analyzing algorithm performance highlights the critical role played by the choice of test instances, and the importance of ensuring diversity and unbiasedness of test instances to support valid conclusions.

Keywords Personnel scheduling · Combinatorial optimization · Algorithm selection · Instance space

Mathematics Subject Classification (2010) 68T05 · 68T20 · 68W40

Financial support from the Austrian Federal Ministry for Digital and Economic Affairs and the National Foundation for Research, Technology and Development, and the Australian Research Council under grant FL140100012, is gratefully acknowledged.

✉ Lucas Kletzander
lkletzan@dbai.tuwien.ac.at

Nysret Musliu
musliu@dbai.tuwien.ac.at

Kate Smith-Miles
smith-miles@unimelb.edu.au

[1] Christian Doppler Laboratory for Artificial Intelligence and Optimization for Planning and Scheduling, DBAI, TU Wien, Vienna, Austria

[2] School of Mathematics and Statistics, University of Melbourne, Parkville, Australia

1 Introduction

A wide variety of algorithms have been proposed to tackle various personnel scheduling problems, due to the practical importance in many professions of scheduling employees in shift work rosters. Such rosters must minimise costs while simultaneously meeting occupational health and safety requirements for employee health and well-being. In any field where a variety of solution methods and algorithms have been proposed, it becomes important to understand how the strengths and weaknesses of algorithms depend on the characteristics of the test instances. This deeper understanding of what makes certain problem instances easy or hard for various solution methods is critical to ensure the best approach will be deployed, rather than selecting the method that performs best on average across an arbitrary test of test benchmark instances.

In order to explain what causes hardness in instances and which algorithms work best on different parts of the instance space, we need a diverse set of instances, features that are able to explain the problem hardness and a representation of the instances where algorithms can be compared in a meaningful way [23]. In this work we aim to achieve such insights for a personnel scheduling problem using a recently developed methodology known as Instance Space Analysis (ISA) [13, 21].

Our focus is the Rotating Workforce Scheduling Problem (RWS), which is a real-life problem well studied in literature with several state-of-the-art solving methods. Although various instances were available, a detailed analysis of those instances as well as strengths and weaknesses of different methods on different instances has not been done before. Such results are both relevant in practice and also provide theoretical insights about the problem and solution method suitability for various instance properties. We first propose a set of features aiming to capture properties describing instance difficulty required for algorithm selection and Instance Space Analysis. We then perform ISA using the toolkit MATILDA[1], revealing that the original set of test instances are not as diverse as hoped, with gaps in the instance space suggesting the opportunity to extend the original set of 2000 instances for the RWS problem to become more diverse and comprehensive. Subsequently, we generate a set of new instances that fill these gaps, and an extended analysis is performed on a this more demonstrably diverse set of test instances.

A feature selection process followed by a dimensional reduction method is performed to represent all the test instances in a $2D$ visualization of the instance space. Inspecting the performance of two different exact models and a metaheuristic approach across this instance space reveals uniquely strong and weak areas in the instance space as well as a complementary coverage of the instance space when combining the strengths of the algorithms. Further we are able to observe a phase transition from feasible to infeasible instances in the instance space, with a more challenging area of instances at this phase transition boundary.

Section 2 discusses related work for RWS and ISA. In Section 3 we describe the RWS problem in more detail and introduce the existing problem instances. Section 4 presents two constraint programming models and a metaheuristic approach for the problem. Section 5 presents the features of the instances that complete the construction of the meta-data required for ISA. In Section 6 we present the first instance space based on our original set of benchmark test instances, before exploring the instance space to understand where additional test instances would add significant value to the analysis. After generating additional instances based on this insight, the extended instance space is then presented and analyzed

[1] https://matilda.unimelb.edu.au

to gain further insights into model suitability for different types of instances. Conclusions and opportunities for future work are identified in Section 7.

2 Related work

This section provides previous work for both the problem we are applying the analysis on and the methodology of Instance Space Analysis.

2.1 Rotating workforce scheduling

The Rotating Workforce Scheduling (RWS) problem is an employee scheduling problem that can be classified as a single-activity tour scheduling problem with non-overlapping shifts and rotation constraints [1, 19] and is known to be NP-complete [4].

So far the problem has been addressed with a range of different methods. Complete approaches include a network flow formulation [2], integer linear programming [10], several constraint programming formulations [9, 11, 16, 25] and an approach with satisfiability modulo theories [5]. There is also work on heuristic approaches [14], including the metaheuristic we use for our evaluation [15], further the creation of rotating schedules by hand [9], and using algebraic methods [6].

A new constraint model using a formulation in MiniZinc was introduced by [17] and evaluated using the lazy clause generation solver Chuffed and the MIP solver Gurobi. [8] extends the direct model in several ways, providing additional constraints to improve handling of infeasible instances and investigating complex real-world constraints and optimization goals. Results are evaluated using Chuffed. We use the models with additional constraints (`EXT1` and `EXT2`) from this work for our evaluation regarding exact methods.

2.2 Instance space analysis

Instance Space Analysis is a methodology developed by Smith-Miles and co-workers [12, 21, 22] in recent years, by extending the Algorithm Selection Problem framework of Rice [20, 24]. Instances are represented as a feature vector that captures the intrinsic difficulty of instances for various algorithms (or models or parameter settings). By constructing a $2D$ projection of a feature-vector representation of instances, Instance Space Analysis (ISA) allows us to:

1. visualize the distribution and diversity of existing benchmark instances;
2. assess the adequacy of the features;
3. identify and measure the algorithm's regions of strength *footprint* and weaknesses; and
4. distinguish areas of the space where it may be useful to generate additional instances to support greater insights.

Figure 1 illustrates the framework and its component spaces. The first is the ill-defined *problem space*, $\mathcal{P}$, which contains all the relevant problems to be solved. However, we only have computational results for a subset, $\mathbf{I}$. Second is the algorithm space, $\mathcal{A}$, which is composed of a portfolio of successful algorithms for the problems in $\mathbf{I}$. Third is the performance space, $\mathcal{Y}$, which is the set of feasible values of $y(\alpha, x)$, a measure of the performance of an algorithm $\alpha \in \mathcal{A}$ to solve a problem $x \in \mathbf{I}$. Fourth is the *feature space*, $\mathcal{F}$, which

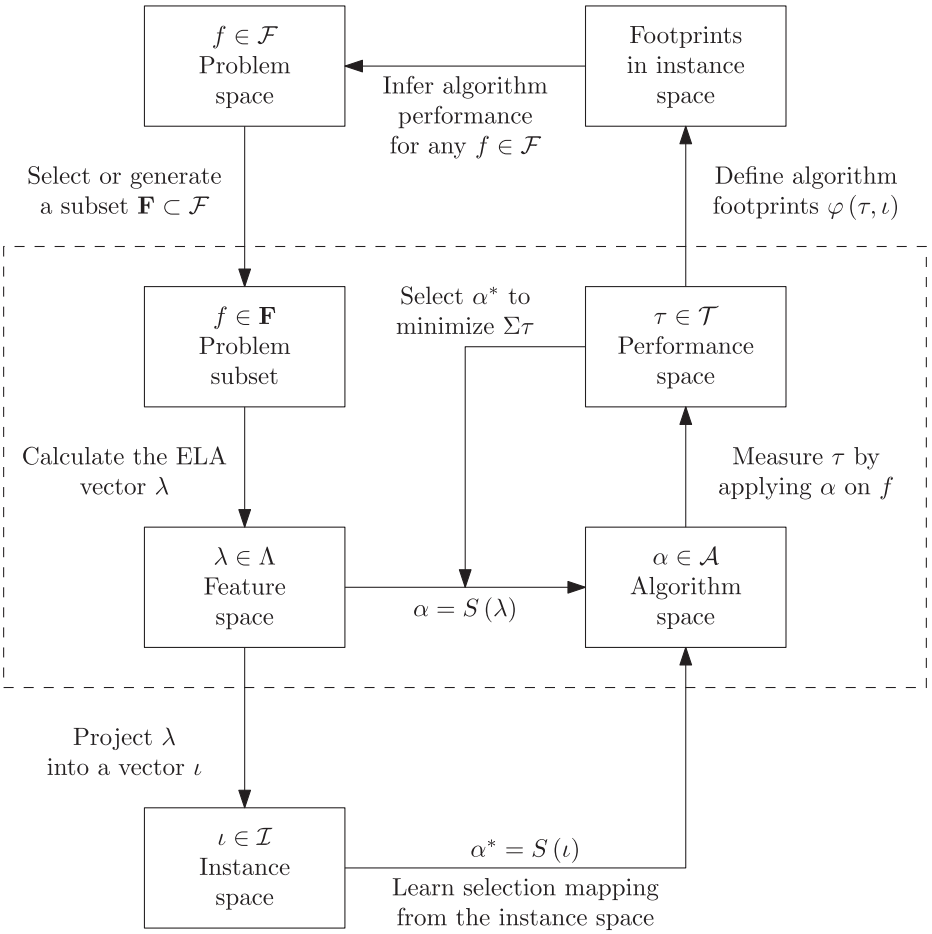


Fig. 1 Algorithm selection framework by [21], which extends the original by [20]

contains multiple measures that characterize the properties that make an instance in **I** difficult. These measures are represented by the vector $\mathbf{f}(x)$. The meta-data, composed of the features and algorithm performance for all the instances in **I**, is used to learn the mapping $g(\mathbf{f}(x), y(\alpha, x))$ that projects an instance x from a high-dimensional feature space to a two-dimensional space, which we call the *instance space*. In earlier work, this projection was achieved using principal component analysis, and applied to problems as diverse as graph coloring [21], time series forecasting [7], and software test case generation methods [18]. In this paper, we adopt the latest version of the evolving methodology described in [13], applied to machine learning algorithms, where a customized projection algorithm was developed to obtain an optimal projection that aims to expose linear trends in both features and algorithm performance to aid interpretability.

While the ISA methodology is broadly applicable, it needs to be customised through careful choice of instance features and an understanding of what makes the problem hard [23].

3 Problem space

A rotating workforce schedule consists of the assignment of shifts or days off to each day across several weeks for a certain number of employees. Table 1 shows an example for four employees (or four equal-sized groups of employees), assigning the three shift types day shift (D), afternoon shift (A), and night shift (N). Each employee starts their schedule in a different row, moving from row i to row $i \bmod n + 1$ (where n is the number of employees) in the following week.

3.1 Problem specification

We use definitions and notation by [16] and [8]. We define:

- n: Number of employees.
- w: Length of the schedule, typically $w = 7$ as the demands repeat in a weekly cycle. The total length of the planning period is $n \cdot w$, as each employee rotates through all n rows.
- $\mathbf{A}$: Set of work shifts (activities), enumerated from 1 to m, where m is the number of shifts. A day off is denoted by a special activity O with numerical value 0 and we define $\mathbf{A}^+ = \mathbf{A} \cup \{O\}$.
- R: Temporal requirements matrix, an $m \times w$-matrix where each element $R_{i,j}$ corresponds to the number of employees that need to be assigned shift $i \in \mathbf{A}$ at day j. The number of employees o_j that need to be assigned a day off on day j can be calculated by $o_j = n - \sum_{i=1}^{m} R_{i,j}$.
- ℓ_w and u_w: Minimal and maximal length of blocks of consecutive work shifts.
- ℓ_s and u_s: Minimal and maximal lengths of blocks of consecutive assignments of shift s given for each $s \in \mathbf{A}^+$.
- Forbidden sequences of shifts: Any sequences of shifts (like $N\ D$, a night shift followed by a day shift) that are not allowed in the schedule. This is typically required due to legal or safety concerns. In practice in is usually sufficient to forbid sequences of length 2 or sequences of length 3 where the middle shift is a day off. These are also the kind of restrictions used in the benchmark instances for Rotating Workforce Scheduling.

The task is to construct a cyclic schedule S, represented as an $n \times w$-matrix, where each $S_{i,j} \in \mathbf{A}^+$ denotes the shift or day off that employee i is assigned during day j in the first period of the cycle. The schedule for employee i through the whole planning period consists of the cyclic sequence of all rows of S starting with row i.

3.2 Instances

For initial evaluation we took all 2000 instances from a standard benchmark set[2]. These instances include 20 real life instances and other randomly generated instances [14, 15, 17]: The instances generated consist of 9 to 51 employees, 2 to 3 shift types, 3 to 4 minimal and 5 to 7 maximal length of work blocks, 1 to 2 minimal and 2 to 4 maximal length of days-off blocks, and minimal and maximal length of periods of consecutive shifts (D: 2 to 3 and 5 to 7, A: 2 to 3 and 4 to 6, N: 2 to 3 and 4 to 5). The same forbidden sequences as for real-life examples are used. Initially the temporal requirements for shifts are distributed randomly

[2] http://www.dbai.tuwien.ac.at/staff/musliu/benchmarks/

Table 1 Example schedule for 4 employees

Empl.	Mon	Tue	Wed	Thu	Fri	Sat	Sun
1	D	D	D	D	N	N	–
2	–	–	A	A	A	A	N
3	N	N	–	–	D	D	D
4	A	A	N	N	–	–	–

between shifts based on the total number of working days and days-off (the number of days-off is set to $\lfloor n \times w \times 0.2857 \rfloor$). With probability 0.3 the temporal requirements during the weekend are changed (half of these duties are distributed to the temporal requirements of the weekdays).

In this paper in the process of Instance Space Analysis we generated additional 4000 instances based on the generator for the first 2000 instances. The generator was modified to generate instances with a larger number of employees and more diverse distribution of workforce requirements during weekdays. As we will explain in later sections these new instances enabled us to cover a larger space of instances. The new instances are available online.[3]

4 Algorithm space

In this section we present the two exact models and the metaheuristic used for the evaluation.

4.1 Constraint models

For the evaluation of an exact approach to RWS we use two different constraint models implemented in the MiniZinc modelling language as presented in [8], based on [17]. This section will give a short summary of the used models, for details refer to the paper.

The models use a constraint for the sum of requirements for each day and shift type, as well as a redundant constraint for days off duty. Block lengths are modelled as follows: Each time a block starts, the next shifts are constrained to be part of the block until the minimum length and at least when reaching the maximum length a different shift assignment has to occur. Some constraints are included for symmetry breaking. A global count constraint is used to enforce boundaries for the number of blocks (see (4) and (5) in the next section).

The difference between the two models `CHUFFED1` and `CHUFFED2` (corresponding to `EXT1` and `EXT2` in [8]) is a set of additional constraints in `CHUFFED2` that enforce bounds for the number of remaining blocks on and off work on the start of each new block across the whole schedule. While this is an effort increasing the number of constraints, for certain instances this might be vital in reducing the search space.

[3] https://cdlab-artis.dbai.tuwien.ac.at/papers/isa-rws/

The evaluation from [8] showed good results for both models, but a considerable increase when using the best of both. Therefore, we expect interesting insights analysing the comparison of the models.

4.2 Metaheuristic solver

A metaheuristic solver for RWS was developed in [14, 15]. This solver includes several methods based on tabu search, min-conflicts heuristic and methods that combine min-conflicts heuristic with tabu mechanism and random walk. The hybrid methods improved the performance of the commercial system for generation of rotating workforce schedules and are currently state of the art heuristic methods for this problem.

5 Features

In order to apply Instance Space Analysis, a set of features needs to be defined. These features need to be able to explain the hardness of instances and the different performances of algorithms to obtain useful results from the analysis. While a subset of the features is eventually used for the instance space, at first a larger set of potentially useful features is created, using three categories.

First we use direct instance parameters or their minimum and maximum values. Second, we contribute new features calculated from the instance specification. Further we obtain numbers of variables and constraints using the conversion of the MiniZinc model as well as the initialization of Chuffed. We explain both the calculation of the features and the reason for including them as well as expectations for their usefulness. In total we present 38 potential features.

5.1 Direct instance features

While not necessarily expressive enough to explain instance hardness on their own, we include 13 features based on instance parameters.

- Number of employees n: This feature is expected to correlate with hardness is some way as larger n requires larger solutions.
- Number of shifts m: While instances with larger numbers of shifts are also expected to increase difficulty, the instances in the existing benchmarks focus on 2 or 3 shifts and still provide a wide range of difficulty. Therefore we keep the investigation in this area regarding the number of shifts.
- Minimum and maximum length of work blocks ℓ_w and u_w as well as blocks off shift ℓ_O and u_O.
- Minimum, maximum and average for each of the sets $\{\ell_s \mid s \in \mathbf{A}\}$ and $\{u_s \mid s \in \mathbf{A}\}$.
- Number of forbidden sequences f.

5.2 Advanced instance features

We select the following 14 new features based on observations working with the algorithms that might help to explain instance hardness.

– *workFraction* (2): Percentage of all days spent working. Equation (1) defines the sum of all requirements.

$$r = \sum_{i=1}^{m} \sum_{j=1}^{w} R_{i,j} \tag{1}$$

$$workFraction = \frac{r}{w \cdot n} \tag{2}$$

– Minimum and maximum of *shiftFraction* (3): Distribution of requirements between shifts.

$$shiftFraction = \left\{ \frac{\sum_{j=1}^{w} R_{s,j}}{w \cdot n} \;\middle|\; s \in \mathbf{A} \right\} \tag{3}$$

– *blockTightness* (6): (4) defines a lower bound for the total number of working blocks in the solution. As the problem is cyclic, the number of work blocks and free blocks needs to be equal, therefore the maximum of the lower bounds for both types is calculated. Equation (5) does the same for the upper bound. This feature can be used to identify instances as infeasible when the value is negative. This fact is used in the constraint models. Further, low positive values seem to lead to harder instances according to the experience in preliminary testing. This might be explained by the reduced possibilities in choosing block lengths for a valid solution.

$$low = \max \left\{ \left\lceil \frac{r}{u_w} \right\rceil, \left\lceil \frac{n \cdot w - r}{u_O} \right\rceil \right\} \tag{4}$$

$$up = \min \left\{ \left\lfloor \frac{r}{\ell_w} \right\rfloor, \left\lfloor \frac{n \cdot w - r}{\ell_O} \right\rfloor \right\} \tag{5}$$

$$blockTightness = up - low \tag{6}$$

– *minAvgBlockLength* (7) and *maxAvgBlockLength* (8): Lower and upper bound for the average block length (work block + consecutive free block) as a different way to use *blockTightness*.

$$minAvgBlockLength = \frac{w \cdot n}{up} \tag{7}$$

$$maxAvgBlockLength = \frac{w \cdot n}{low} \tag{8}$$

– Minimum and maximum of *shiftBlockTightness* (11): Freedom in choosing block lengths for individual shift types. Equations (9) and (10) calculate lower and upper bounds for the number of blocks for each type $s \in \mathbf{A}$.

$$low_s = \left\lceil \frac{\sum_{j=1}^{w} R_{s,j}}{u_s} \right\rceil \tag{9}$$

$$up_s = \left\lfloor \frac{\sum_{j=1}^{w} R_{s,j}}{\ell_s} \right\rfloor \tag{10}$$

$$shiftBlockTightness = \{up_s - low_s \mid s \in \mathbf{A}\} \tag{11}$$

– Minimum and maximum of *shiftDayFactor* (12): Regularity of shifts throughout the week.

$$shiftDayFactor = \left\{ \frac{\min\{R_{s,j} \mid j \in \mathbf{W}\}}{\max\{R_{s,j} \mid j \in \mathbf{W}\}} \;\middle|\; s \in \mathbf{A} \right\} \tag{12}$$

– Minimum and maximum of *dayFraction* (13): Workload in relation to the number of employees for individual days.

$$dayFraction = \left\{ \frac{\sum_{i=1}^{m} R_{i,j}}{n} \;\middle|\; j \in \mathbf{W} \right\} \tag{13}$$

– Minimum and maximum of *dailyChange* (14): Change in workload between consecutive days.

$$dailyChange = \left\{ \sum_{i=1}^{m} R_{i,j+1} - \sum_{i=1}^{m} R_{i,j} \right\} \tag{14}$$

5.3 Model features

Further we use the standard constraint model of RWS modelled in MiniZinc and the initialization in the lazy clause generation solver Chuffed to obtain numbers of variables and constraints for 11 further features.

MiniZinc first converts the model into a low-level format called FlatZinc. From the MiniZinc to FlatZinc conversion statistics we obtain the following four features about the FlatZinc model of the respective instance:

– Number of boolean variables
– Number of integer variables
– Number of boolean constraints
– Number of integer constraints

From the initialization of the instance in Chuffed we obtain the following seven features taken from the solver representation of the problem:

– Number of variables
– Number of propagators
– Number of SAT variables
– Number of binary clauses
– Number of ternary clauses
– Number of long clauses
– Average length of long clauses

There could be an advantage in such features as they are rather problem independent, which might allow transferring results to other problems. On the other hand, these features might not capture the structure of the instances well enough to be useful.

5.4 Initial experiments

While we could immediately try ISA with our features, we decided to start with experiments using classical machine learning techniques on a test run comparing the base constraint model with the metaheuristic on the original set of 2000 instances using a timeout of 200 seconds. These are meant to give us a good impression whether our features are helpful in learning. Table 2 provides results using Random Forests [3] for various predictions. The result column reports accuracy (classification) or correlation coefficient (regression). The results were obtained using Weka[4] with the default settings and 10 fold cross validation.

[4] https://www.cs.waikato.ac.nz/~ml/weka/index.html

Table 2 Results for various predictions using Random Forests

Target	Features	Result
Best method	all	76.15%
Best method	no direct	75.55%
Best method	no advanced	70.90%
Best method	no model	77.85%
Solved	all	92.35%
Solved	no direct	90.90%
Solved	no advanced	88.70%
Solved	no model	92.40%
Feasible	all	97.97%
Feasible	no direct	97.28%
Feasible	no advanced	91.96%
Feasible	no model	98.35%
log(runtime Chuffed)	all	0.855
log(runtime Chuffed)	no direct	0.839
log(runtime Chuffed)	no advanced	0.766
log(runtime Chuffed)	no model	0.866
log(runtime Meta)	all	0.878
log(runtime Meta)	no direct	0.856
log(runtime Meta)	no advanced	0.788
log(runtime Meta)	no model	0.883

For the best method, 3 classes are possible (Chuffed, metaheuristic, tie). Solved (boolean) corresponds to all instances that could be solved by either Chuffed or the metaheuristic without timeout. Feasible (boolean) only uses instances which could be solved by either method. Regarding the runtime, the distribution shows many instances with very short runtime, fewer with longer runtimes and a number of timeouts. Therefore, predicting the magnitude of the runtime worked much better.

The prediction results are overall promising that we have meaningful features to proceed with ISA. While the results on predicting the best method are below 80%, for the prediction of hard (not solved by any method) or feasible instances we can achieve more than 90% accuracy. For the runtimes, the result shows the correlation coefficient exceeding 0.85.

Regarding the features, we compared using all 38 features to skipping either the direct, advanced, or model features. The results show for all predictions that the best results are obtained not using all features, but excluding the model features, while skipping the advanced features leads to the worst results. Therefore, we are confident that the advanced features are useful for ISA and proceed with 27 features, excluding the model features.

6 Instance space analysis

We perform the Instance Space Analysis using the Matlab toolkit MATILDA available from https://matilda.unimelb.edu.au. It uses a configuration file to specify parameters for the

analysis and then performs the following steps. It bounds extreme outliers and does normalization using a Box-Cox and Z transformation. Next, features with low diversity are removed. Features with high correlation to performance are retained and a clustering step is applied. Then it calculates the projection to the 2-dimensional instance space and the footprints of the algorithms within the space.

The algorithms are executed on an Intel Core i7-7500 CPU with 2.7 GHz and 16 GB RAM using a timeout of 1000 seconds. The metaheuristic is executed three times on each instance to account for some variation in the runtimes. While there are some outliers where the algorithm reaches a solution particularly early or late, for the vast majority of instances the distinction between the runtimes of the Chuffed models and the metaheuristic is large enough that such variations do not influence which algorithm is the fastest.

The settings for MATILDA are mostly the default settings, using all processing steps. The performance metric is set to the runtime (lower is better). The diversity threshold is reduced to 0.01 (eliminate features where the number of unique values is less than 1% of the number of instances), as some integer features do not have high diversity, but might still be interesting for the evaluation, e.g., *blockTightness*. The correlation threshold is 5, the silhouette threshold for clustering is 0.45.

An instance space analysis provides a framework to scrutinize the diversity of test instances, and to identify opportunities to augment a test suite to ensure absence of bias and maximal diversity across real-world and theoretical instance types. While it may be possible to analyze real-world instances and note their difference to synthetically generated instances, in practice it can be challenging to modify synthetic generators to ensure they improve their diversity and real-world-likeness. A visualization of the instances helps provide confidence that additional instances are similar to real-world instances, and are also simultaneously adding value to the diversity of the test suite.

In the following we first report results for the original instances and discuss our conclusions from these results. Then we evaluate the extended set of instances generated based on those conclusions and discuss the new insights we get.

6.1 Original instances

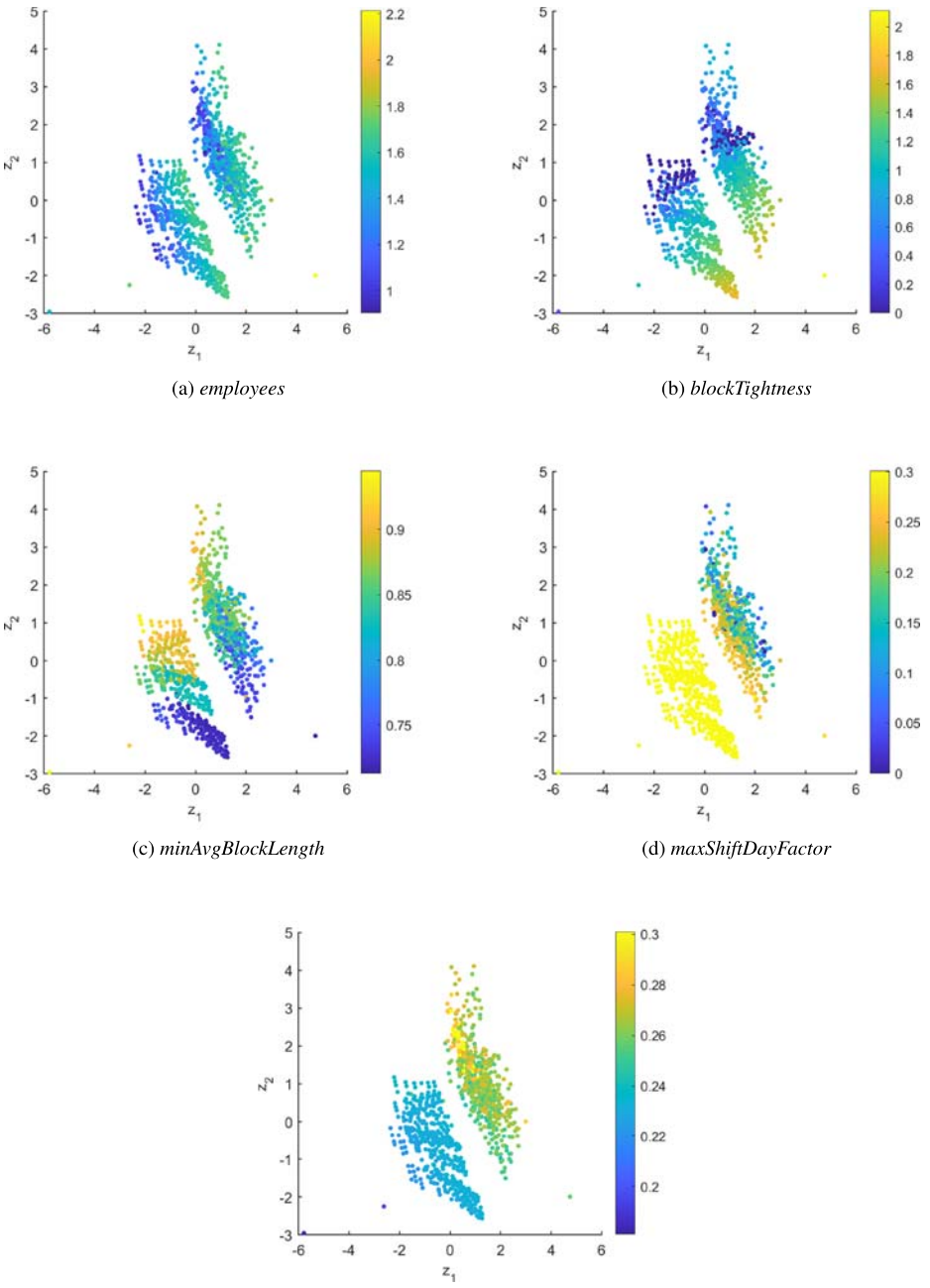
For the first set of experiments we used the existing dataset of 2000 instances. We evaluate the results from the base constraint model as well as the metaheuristic.

6.1.1 Instance distribution

Figure 2 shows the resulting instance space and the distribution of the selected features. Note that values on the scale correspond to normalized feature values. Equation (15) shows the projection matrix applied to the Box-Cox transformed and normalized feature values.

$$\begin{pmatrix} z_1 \\ z_2 \end{pmatrix} = \begin{pmatrix} -0.45 & -0.39 \\ 0.45 & 0.40 \\ 0.50 & 0.08 \\ -0.32 & 0.37 \\ 0.23 & -0.63 \end{pmatrix}^{\mathsf{T}} \cdot \begin{pmatrix} maxShiftDayFactor' \\ maxDayFraction' \\ employees' \\ minAvgBlockLength' \\ blockTightness' \end{pmatrix} \tag{15}$$

With *blockTightness* and *minAvgBlockLength*, two features describing possible distributions of block lengths were selected by the analysis. Further, the number of employees representing the size of the instances is present. The other two features represent the distribution throughout the week (*maxShiftDayFactor*) and the daily workload (*maxDayFraction*).



(a) *employees*

(b) *blockTightness*

(c) *minAvgBlockLength*

(d) *maxShiftDayFactor*

(e) *maxDayFraction*

Fig. 2 Selected features for the original instance set

One immediate observation is that the instances cluster in two areas with a separating gap in between. The feature most clearly describing the difference is *maxShiftDayFactor*, where the left cluster almost uniformly has the highest value. In fact, this cluster consists of instances with constant demand on different days of the week while the other cluster has varying demand on some days.

Further, the first 20 instances are not randomly generated, but stem from real-world scenarios. Some of these show as outliers in the results, hinting that the randomly generated instances do not cover all of those scenarios.

6.1.2 Algorithm evaluation

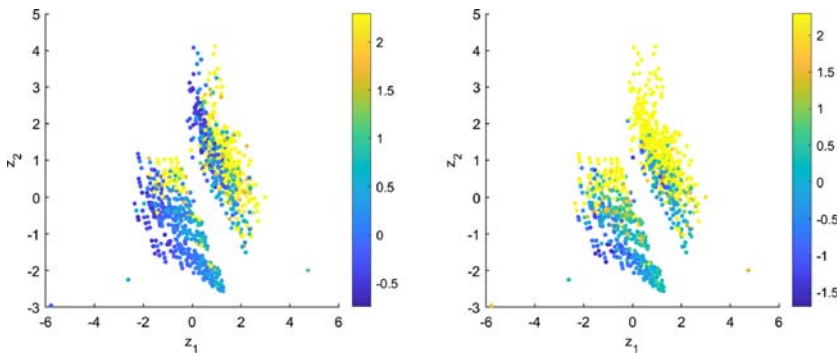
Figure 3 shows the results of the evaluated algorithms on the original instances. The results show two trends. First, instances higher along the z_2 coordinate seem to be harder. This distribution is especially visible for the metaheuristic. A lower z_2 coordinate, on the other hand, corresponds to higher *blockTightness* and lower *minAvgBlockLength*, translating to more possibilities for choosing the length of blocks. Second, within each cluster, instances with higher z_1 coordinate seem to be harder. This mostly corresponds to higher numbers of employees.

6.2 Extended instances

Due to the gaps and outliers explained in the previous section as well as the fact that algorithm performance seems similar in the two clusters of instances, we decided to extend the set of instances with two goals.

First, we want to close the gap between the clusters of instances, mainly by changing the way the instance generator handles the distribution of shifts across days. Second, we want to expand the borders of the clusters in general in order to explore a wider region of the instance space and cover the real life instances more comprehensively.

However, some features like the number of employees are unbounded or may have natural restrictions on values that make sense for practical instances, such as avoiding extremely low demand relative to the number of employees. We increased the maximum number of employees within the generated instances from 51 to 108. For several other generator options we used wider ranges of values than before.



(a) Early Chuffed model (b) Metaheuristic

Fig. 3 Algorithm runtimes for the original instance set

In total we generated 4000 new instances, dropping 520 of those with *blockTightness* lower than the previous minimum, as these instances are infeasible anyway and can be identified rather easily. Together with the original instances, we evaluated 5480 instances for the following results.

6.2.1 Instance distribution

Figure 4 shows the distribution of original and new instances in the new instance space. Note that due to the re-computation of the space with more instances and all three algorithms the selection of features changed slightly. Also, the whole projection is now rotated roughly 90 degrees clockwise from the original projection. The new instances mostly fill the gap between the previous clusters and extend the instances towards the area with high values in both coordinates. Almost all original instances (19 out of 20) are now within the more densely populated areas of the instance space, indicating good coverage of real-world scenarios.

Figure 5 shows the selected features for the extended instance set. The projection matrix is given in (16).

$$\begin{pmatrix} z_1 \\ z_2 \end{pmatrix} = \begin{pmatrix} -0.31 & 0.31 \\ 0.02 & -0.57 \\ -0.47 & -0.08 \\ 0.44 & 0.15 \end{pmatrix}^{\mathsf{T}} \cdot \begin{pmatrix} minDayFraction' \\ maxDayFraction' \\ maxAvgBlockLength' \\ minAvgBlockLength' \end{pmatrix} \tag{16}$$

We see that this time there is no obvious gap between the instances. There is a trail of thinly distributed instances with high values on both coordinates identified by very high

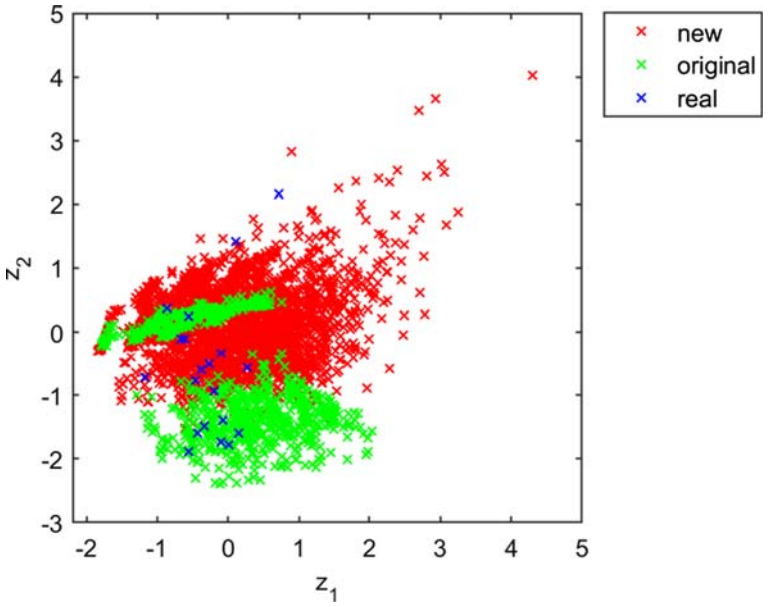
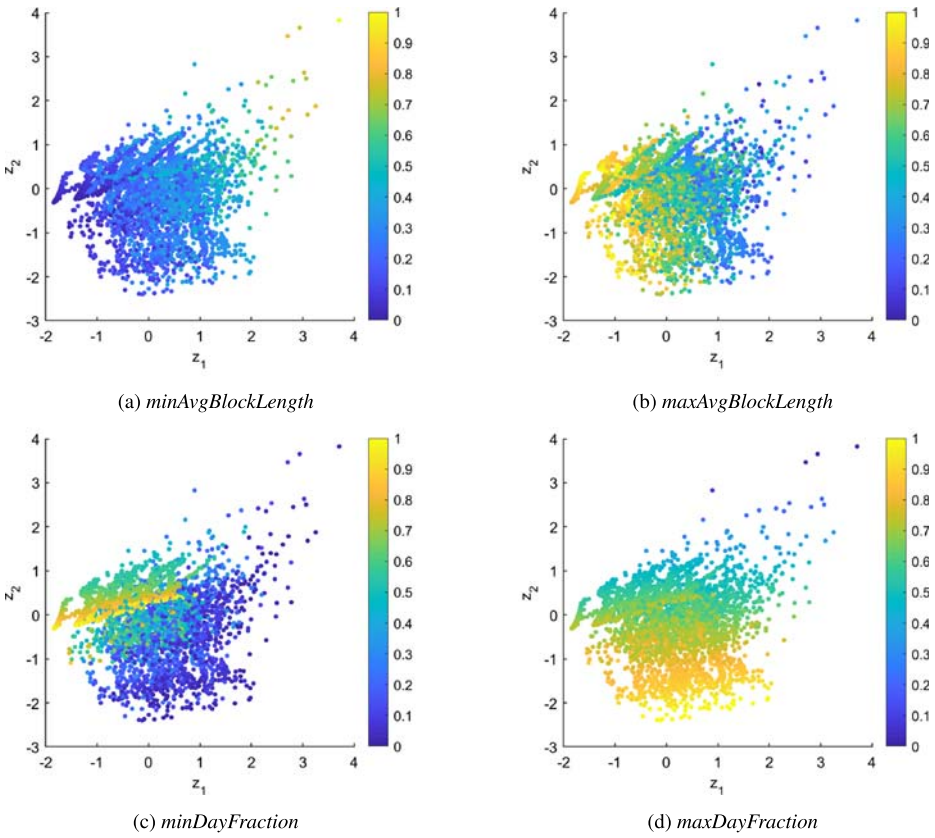


Fig. 4 Instance distribution for the extended instance set



(a) *minAvgBlockLength* (b) *maxAvgBlockLength*

(c) *minDayFraction* (d) *maxDayFraction*

Fig. 5 Selected features for the extended instance set

values of *minAvgBlockLength*. Instances at this extreme of the instance space are not practical (very long blocks of consecutive work days) and are also clearly not feasible as seen in Fig. 7, therefore, we saw no need to create further instances in this region.

Regarding the selected features the representation of *blockTightness* was replaced by *maxAvgBlockLength*. The number of employees is not present any more, even though a wider range of values is used in the new instances, hinting that other factors are more critical for instance hardness in this more diverse dataset. *maxShiftDayFactor*, previously identifying a whole cluster of instances, is not present either. However, its distribution is very similar to the ratio between minimum and maximum of *dayFraction*.

In total, we can identify two main axes in the instance space. In the z_1 direction, we can identify instances with low minimum and high maximum average block length for low z_1 values, while high minimum and low maximum can be found for high z_1 values. Therefore, low z_1 corresponds to a wide range of possibilities for choosing block lengths, while high z_1 corresponds to a narrow range of possibilities or even infeasibility, as seen later.

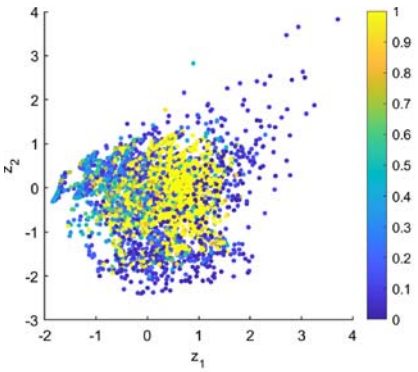
In the z_2 direction, low values identify regions with low minimum and high maximum *dayFraction*. This corresponds to large differences in demand between different days. High values identify regions with high minimum and low maximum *dayFraction*, corresponding to little to no difference between different days.

6.2.2 Algorithm evaluation

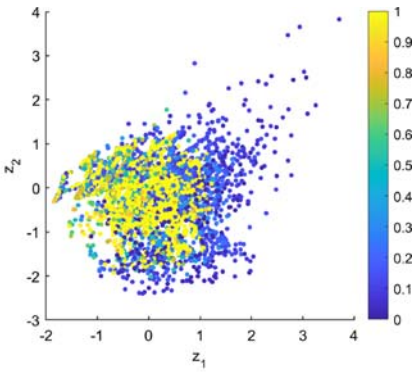
Figure 6 shows the runtimes of the algorithms on the extended instances, again for a timeout of 1000 seconds. We can draw several conclusions from these figures. First, both Chuffed versions are very fast in the low z_2 and high z_1 areas, while the metaheuristic cannot solve instances in this area. Regarding the central area, the Chuffed versions show different distributions. For `CHUFFED1` the bad area is more to the right of the center (higher z_1), while for `CHUFFED2` it is more to the left of the center (lower z_1). The metaheuristic shows fast runtimes in the area of low z_1 and high z_2.

Figure 7 shows the algorithm results in the categories feasible solution (green), proven infeasible (red) and timeout (yellow).

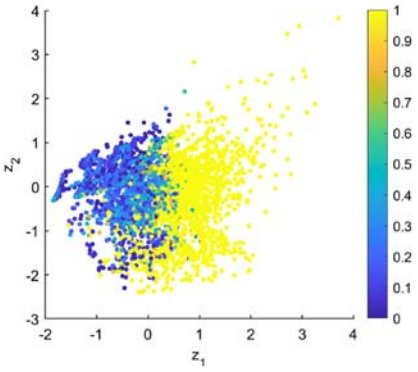
For the results from Chuffed, both versions show mostly feasible results for low z_1 coordinates and mostly infeasible results for high z_1 and low z_2 coordinates. However, we can see clear differences in the distribution of the timeout instances. For `CHUFFED1`, the timeouts are more towards the infeasible side, while for `CHUFFED2`, they are more towards the feasible side. This indicates that `CHUFFED2` is stronger at detecting infeasibility at the cost of lower performance in the feasible area while `CHUFFED1` is stronger in the feasible area.



(a) `CHUFFED1`

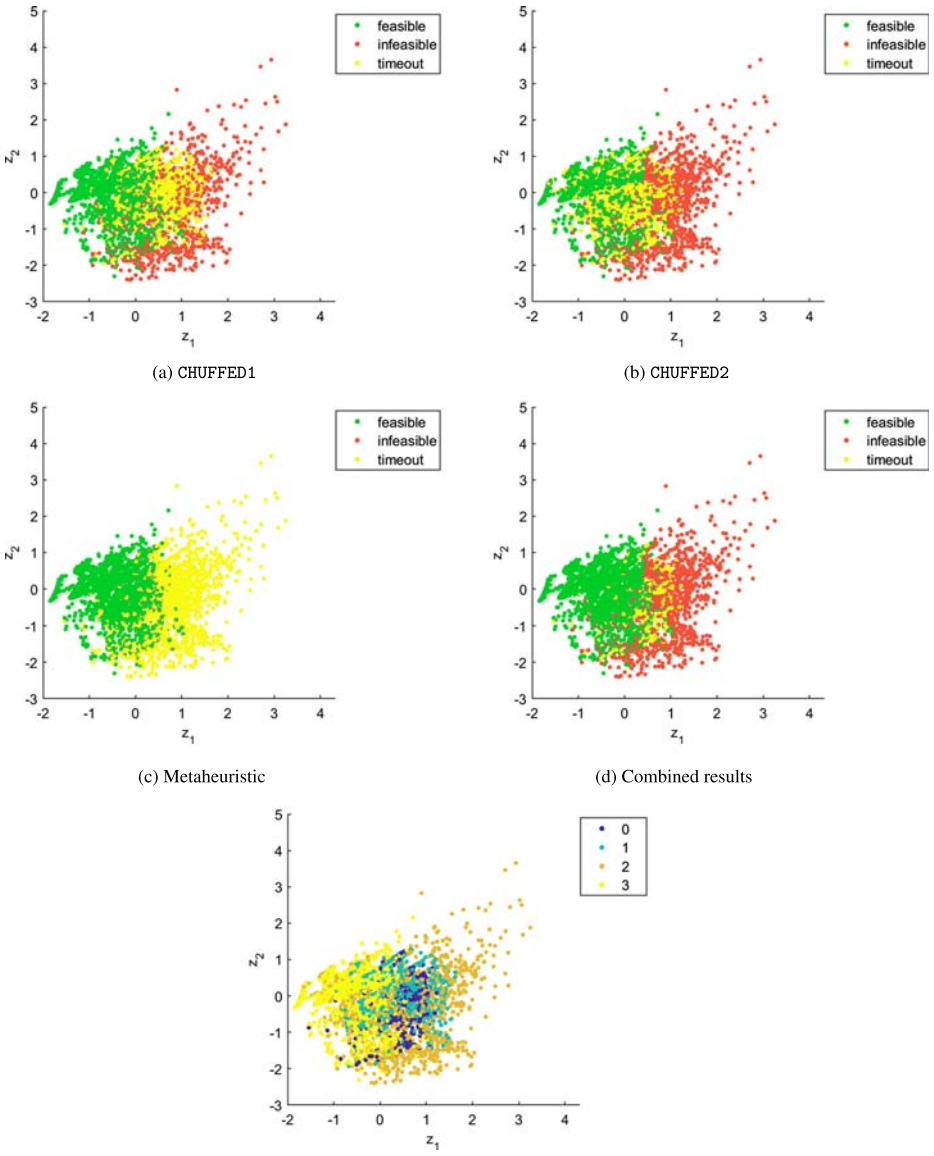


(b) `CHUFFED2`



(c) Metaheuristic

Fig. 6 Algorithm runtimes for the extended instance set



(a) CHUFFED1

(b) CHUFFED2

(c) Metaheuristic

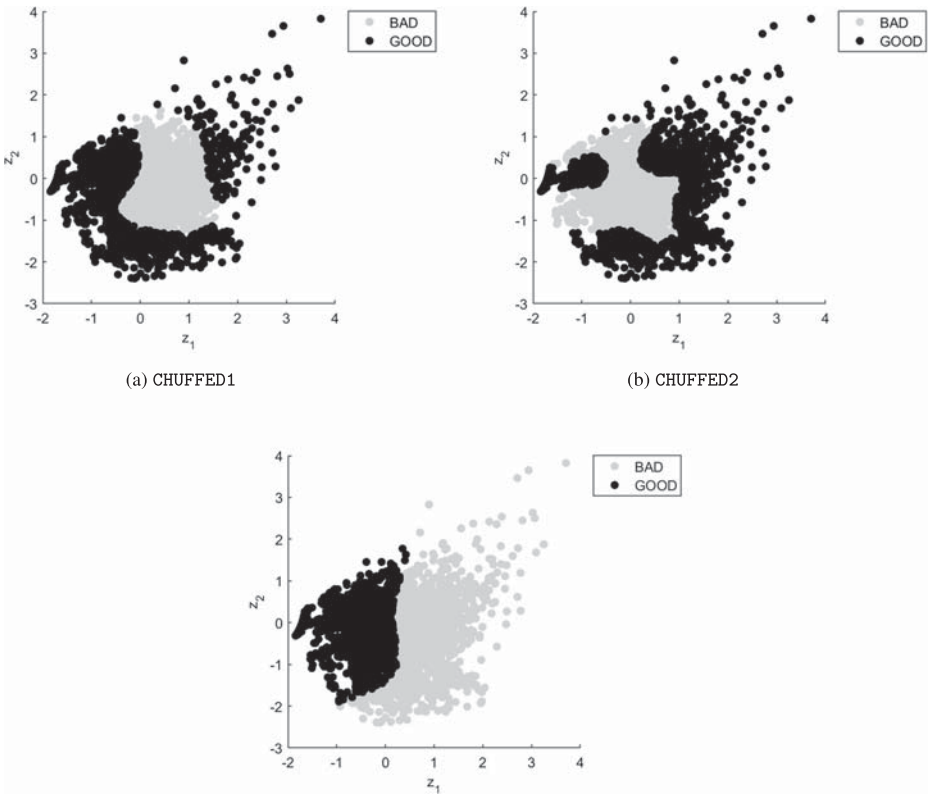
(d) Combined results

(e) Number of obtained results

Fig. 7 Algorithm results for the extended instance set

For the metaheuristic the result is different as it is not able to identify infeasible instances, but rather searches for a result until timeout. In comparison to the combined chart the metaheuristic seems to work very well for most of the feasible instances.

Several conclusions can be drawn from the next charts, showing the distribution of feasible and infeasible instances as well as the number of successful algorithms. The feasibility chart shows instances where at least one algorithm found a feasible solution (green), at least one algorithm proved infeasibility (red), or timeout on all algorithms (yellow). First,



(a) CHUFFED1

(b) CHUFFED2

(c) Metaheuristic

Fig. 8 Algorithm footprints for the extended instance set

we notice a clear distribution of feasible and infeasible instances using our features. Further, there is a low number of unsolved instances, also compared to individual algorithm results, showing that the portfolio of algorithms can combine the different strengths of the algorithms.

Regarding the number of solutions, in the low z_1 region, for most instances all algorithms could find a solution. For high z_1 or low z_2, both Chuffed algorithms could prove infeasibility for most instances. Along the border, however, there is a cluster of instances with a low number of successful algorithms, for several instances actually no algorithm found a solution. This tells us that instances on the border of feasibility are usually harder. Among those, especially instances with medium z_2 are hard. Furthermore, our set of instances covers both the hard border region and easier regions on either side, indicating a good diversity of instances.

Extending the analysis beyond the runtimes of each algorithm across the instance space, we can also visualize the regions where each algorithm is expected to perform well based on generalization of statistical evidence. We start by defining good performance of an algorithm to be reporting a solution or infeasibility within 100 seconds. Each algorithm's performance on each instance is assigned a binary label as "good" or "bad" depending on its runtime and application of this arbitrary goodness criteria, which is user-defined and can be

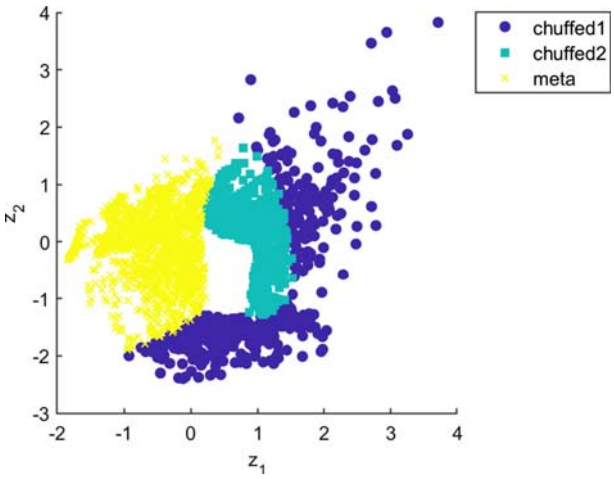


Fig. 9 Algorithm portfolio

varied to explore robustness. Machine learning methods, in this case a support vector machine, is used in MATILDA to learn the statistical boundary between good and bad performance based on the feature vector as input and binary label as output. Further details of the methodology are provided in [13]. Figure 8 shows the resulting algorithm "footprints" as black regions across the instance space for each algorithm. The accuracy of the SVM for the footprints lies at 71.2% for `CHUFFED1`, 65.4% for `CHUFFED2` (corresponding to a more complex shape of the footprint) and 88.1% for the Metaheuristic which has a very clearly defined footprint.

6.2.3 Algorithm portfolio

In order to combine the strengths of the different algorithms, a portfolio approach can be used. Figure 9 shows that for clearly feasible instances the metaheuristic is the best choice, while for the infeasible space and feasible instances close to the feasibility border the Chuffed models are the best choice. `CHUFFED2` is better in the central infeasible area close to the border with medium z_2 values, while `CHUFFED1` is best for the outer areas. For the white area at the center no particular method has a high confidence to succeed. For these instances longer computation times or trying multiple methods at once might be necessary.

The portfolio provides a very clear division of the instance space and an accuracy for selecting the best method (excluding the white center) of 76.4%. Compared to the initial experiments using only one Chuffed model and the Metaheuristic as well as a tie option (timeout on both), this is almost the same accuracy, yet with using an additional algorithm. By providing this representation as part of ISA it not only gives a potential best algorithm, but combined with other data can also help determine whether to expect instances to be easy or hard to solve, or whether they might be infeasible.

7 Conclusion

This paper has used Instance Space Analysis for the first time to evaluate the performance of two constraint models and a metaheuristic technique on the Rotating Workforce Scheduling

problem. We defined a novel set of features and evaluated them by using machine learning with Random Forests to show their predictive capabilities for understanding algorithm performance.

With the results of an initial instance space analysis using the original benchmark dataset, we were able to then produce a larger, more diverse dataset to cover a larger part of the instance space spanning the real life instances and closing gaps in the instance space.

Using the extended dataset, we identified four features that provide a good $2D$ projection for the analysis of the instance space, corresponding to the possibilities for block lengths and the distribution differences between different days. We found different strong and weak areas for the individual algorithms and showed that in combination the algorithms cover the overall instance space well. We also identified areas of feasible and infeasible instances and linked the hardness of instances with the transition between these areas. Furthermore, we were able to generate a portfolio approach that uses the most suitable algorithm based on the location of any instance in the instance space.

Based on these promising results for RWS, in future work we can extend the analysis to consider variations that add additional real world complexities for a range of personnel scheduling problems. The feature set for these extensions will need to be augmented, but we expect that new insights will be possible for these problems if suitable features can be found to capture instance difficulty for various solution methods.

Funding Information Open access funding provided by TU Wien (TUW). The financial support by the Austrian Federal Ministry for Digital and Economic Affairs and the National Foundation for Research, Technology and Development, and the Australian Research Council under grant FL140100012, is gratefully acknowledged.

References

1. Baker, K.R.: Workforce allocation in cyclical scheduling problems: a survey. J. Oper. Res. Soc. **27**(1), 155–167 (1976)
2. Balakrishnan, N., Wong, R.T.: A network model for the rotating workforce scheduling problem. Networks **20**(1), 25–42 (1990)
3. Breiman, L.: Random forests. Mach. Learn. **45**(1), 5–32 (2001)
4. Chuin Lau, H.: On the complexity of manpower shift scheduling. Comput. Oper. Res. **23**(1), 93–102 (1996)
5. Erkinger, C., Musliu, N.: Personnel scheduling as satisfiability modulo theories. In: International Joint Conference on Artificial Intelligence – IJCAI 2017, Melbourne, Australia, August 19-25, 2017, pp. 614–621 (2017). https://doi.org/10.24963/ijcai.2017/86
6. Falcón, R., Barrena, E., Canca, D., Laporte, G.: Counting and enumerating feasible rotating schedules by means of gröbner bases. Math. Comput. Simul. **125**, 139–151 (2016)
7. Kang, Y., Hyndman, R., Smith-Miles, K.: Visualising forecasting algorithm performance using time series instance spaces. Int. J. Forecast **33**(2), 345–358 (2017). https://doi.org/10.1016/j.ijforecast.2016.09.004

8. Kletzander, L., Musliu, N., Gärtner, J., Krennwallner, T., Schafhauser, W.: Exact methods for extended rotating workforce scheduling problems. In: Proceedings of the Twenty-Ninth International Conference on Automated Planning and Scheduling, vol. 29, pp. 519–527. American Association for Artificial Intelligence (AAAI) (2019)
9. Laporte, G.: The art and science of designing rotating schedules. J. Oper. Res. Soc. **50**, 1011–1017 (1999)
10. Laporte, G., Nobert, Y., Biron, J.: Rotating schedules. Eur. J. Oper. Res. **4**(1), 24–30 (1980)
11. Laporte, G., Pesant, G.: A general multi-shift scheduling system. J. Oper. Res. Soc. **55**(11), 1208–1217 (2004)
12. Muñoz, M., Smith-Miles, K.: Performance analysis of continuous black-box optimization algorithms via footprints in instance space. Evol. Comput. **25**(4), 529–554 (2017). https://doi.org/10.1162/EVCO_a_00194
13. Muñoz, M.A., Villanova, L., Baatar, D., Smith-Miles, K.: Instance spaces for machine learning classification. Mach. Learn. **107**(1), 109–147 (2018)
14. Musliu, N.: Combination of local search strategies for rotating workforce scheduling problem. In: International Joint Conference on Artificial Intelligence – IJCAI 2005, Edinburgh, Scotland, UK, July 30 - August 5, 2005, pp. 1529–1530. http://ijcai.org/Proceedings/05/Papers/post-0448.pdf (2005)
15. Musliu, N.: Heuristic methods for automatic rotating workforce scheduling. Int. J. Comput. Intell. Res. **2**(4), 309–326 (2006)
16. Musliu, N., Gärtner, J., Slany, W.: Efficient generation of rotating workforce schedules. Discret. Appl. Math. **118**(1-2), 85–98 (2002)
17. Musliu, N., Schutt, A., Stuckey, P.J.: Solver independent rotating workforce scheduling. In: International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research, pp. 429–445. Springer (2018)
18. Oliveira, C., Aleti, A., Grunske, L., Smith-Miles, K.: Mapping the effectiveness of automated test suite generation techniques. IEEE Trans. Reliab. **67**(3), 771–785 (2018)
19. Restrepo, M.I., Gendron, B., Rousseau, L.M.: Branch-and-price for personalized multiactivity tour scheduling. INFORMS J. Comput. **28**(2), 334–350 (2016)
20. Rice, J.: The algorithm selection problem. In: Advances in Computers, vol. 15, pp. 65–118. Elsevier (1976). https://doi.org/10.1016/S0065-2458(08)60520-3
21. Smith-Miles, K., Baatar, D., Wreford, B., Lewis, R.: Towards objective measures of algorithm performance across instance space. Comput. Oper. Res. **45**, 12–24 (2014). https://doi.org/10.1016/j.cor.2013.11.015
22. Smith-Miles, K., Bowly, S.: Generating new test instances by evolving in instance space. Comput. Oper. Res. **63**, 102–113 (2015). 10.1016/j.cor.2015.04.022
23. Smith-Miles, K., Lopes, L.: Measuring instance difficulty for combinatorial optimization problems. Comput. Oper. Res. **39**(5), 875–889 (2012)
24. Smith-Miles, K.A.: Cross-disciplinary perspectives on meta-learning for algorithm selection. ACM Computing Surveys (CSUR) **41**(1), 6 (2009)
25. Triska, M., Musliu, N.: A constraint programming application for rotating workforce scheduling. In: Developing Concepts in Applied Intelligence, Studies in Computational Intelligence, vol. 363 , pp. 83–88. Springer, Berlin (2011)