Instance Space Analysis for a Personnel Scheduling Problem

Lucas Kletzander[1] , **Nysret Musliu**[1] and **Kate Smith-Miles**[2]
[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, Australia
{lkletzan, musliu}@dbai.tuwien.ac.at, smith-miles@unimelb.edu.au

Abstract

In order to analyse strengths and weaknesses of different solution methods, we use Instance Space Analysis to perform an in-depth evaluation on the Rotating Workforce Scheduling Problem. At first we present a set of features aiming to describe hardness of instances. We create a new, more diverse set of instances based on a first analysis revealing gaps and possible extensions. The results of three algorithms on the extended instance set show different strong and weak areas in the instance space as well as a transition from feasible to infeasible instances including a more challenging area of instances at this transition.

1 Introduction

Due to the need for employees to work in various shifts in many professions, different algorithms are developed and evaluated for a wide range of personal scheduling problems. However, for a detailed analysis of strengths and weaknesses of solution methods, as well as for a deeper understanding what makes certain problem instances easy or hard, it is not sufficient to just compare average results on arbitrary benchmarks.

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. This is done using Instance Space Analysis (ISA).

In this paper we consider the Rotating Workforce Scheduling Problem (RWS) and present a set of features aiming to describe the information required for algorithm selection and instance space analysis. We perform ISA using the toolkit MATILDA, revealing gaps and possible extensions for the original set of 2000 instances for the problem. Therefore, new instances are created and extended analysis is performed on a more diverse selection of instances.

We present the selection of features for the representation of the instance space. The results for two different exact models and a metaheuristic approach show different strong and weak areas in the instance space as well as a good coverage of the instance space when combining the strengths of the algorithms. Further we describe a transition from feasible to infeasible instances in the space and a more challenging area of instances at this transition.

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 problem is an employee scheduling problem that can be classified as a single-activity tour scheduling problem with non-overlapping shifts and rotation constraints [Baker, 1976; Restrepo *et al.*, 2016] and is known to be NP-complete [Chuin Lau, 1996].

So far the problem has been addressed with a range of different methods. Complete approaches include a network flow formulation [Balakrishnan and Wong, 1990], integer linear programming [Laporte *et al.*, 1980], several constraint programming formulations [Laporte, 1999; Musliu *et al.*, 2002; Laporte and Pesant, 2004; Triska and Musliu, 2011] and an approach with satisfiability modulo theories [Erkinger and Musliu, 2017]. There is also work on heuristic approaches [Musliu, 2005], including the metaheuristic we use for our evaluation [Musliu, 2006], further the creation of rotating schedules by hand [Laporte, 1999], and using algebraic methods [Falcón *et al.*, 2016].

A new constraint model using a formulation in MiniZinc was introduced by [Musliu *et al.*, 2018] and evaluated using the lazy clause generation solver Chuffed and the MIP solver Gurobi. [Kletzander *et al.*, 2019] 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 [Smith-Miles *et al.*, 2014; Smith-Miles and Bowly, 2015; Muñoz and Smith-Miles, 2017] in recent years, by extending the Algorithm Selection

Problem framework of Rice [Rice, 1976; Smith-Miles, 2009]. 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 2-d 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 contains multiple measures that characterize the properties that make an instance in $\mathbf{I}$ 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 $\mathbf{I}$, 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 [Smith-Miles *et al.*, 2014], time series forecasting [Kang *et al.*, 2017], and software test case generation methods [Oliveira *et al.*, 2018]. In this paper, we adopt the latest version of the evolving methodology described in [Muñoz *et al.*, 2018], 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.

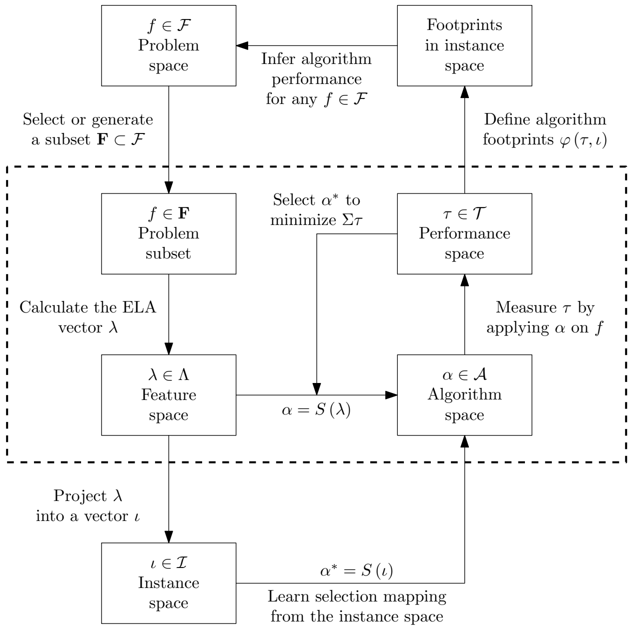
While the ISA methodology is broadly applicable, it needs to be customised through careful choice of instance features and an understanding of what makes the problem hard [Smith-Miles and Lopes, 2012].

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 mod $n + 1$ (where n is the number of employees) in the following week.

3.1 Problem Specification

We use definitions and notation by [Musliu *et al.*, 2002] and [Kletzander *et al.*, 2019]. We define:



$f \in \mathcal{F}$ Problem space
Footprints in instance space
Infer algorithm performance for any $f \in \mathcal{F}$
Select or generate a subset $\mathbf{F} \subset \mathcal{F}$
Define algorithm footprints $\varphi(\tau, \iota)$
$f \in \mathbf{F}$ Problem subset
Select α^* to minimize $\Sigma\tau$
$\tau \in \mathcal{T}$ Performance space
Calculate the ELA vector λ
Measure τ by applying α on f
$\lambda \in \Lambda$ Feature space
$\alpha = S(\lambda)$
$\alpha \in \mathcal{A}$ Algorithm space
Project λ into a vector ι
$\iota \in \mathcal{I}$ Instance space
$\alpha^* = S(\iota)$
Learn selection mapping from the instance space

Figure 1: Algorithm selection framework by [Smith-Miles *et al.*, 2014], which extends the original by [Rice, 1976].

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	-	-	-

Table 1: Example schedule for 4 employees

- 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.
- 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[1]. These instances include 20 real life instances and other randomly generated instances [Musliu, 2006; Musliu, 2005; Musliu *et al.*, 2018]: 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 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 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.

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 [Kletzander *et al.*, 2019]. 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 equations (4) and (5) in the next section).

The difference between the two models `CHUFFED1` and `CHUFFED2` (corresponding to `EXT1` and `EXT2` in [Kletzander *et al.*, 2019]) 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.

The evaluation from [Kletzander *et al.*, 2019] 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 [Musliu, 2005; Musliu, 2006]. 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 add new parameters calculated from the instance specification. Further we obtained numbers of variables and constraints using the conversion of the MiniZinc model as well as the initialization of Chuffed for 11 further features. However, when we trained some preliminary models to perform algorithm selection, we found that the model and solver features did not improve performance. To the contrary, without these features, the performance was slightly better. Therefore, for further analysis, we dropped these features and used the direct and advanced instance features, in total a set of 27 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 features is expected to correlate with hardness is some way as larger n requires larger solutions.
- Number of shifts m: While instances with larger 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 $\{u_s \mid s \in \mathbf{A}\}$.

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

- Number of forbidden sequences f.

5.2 Advanced Instance Features

We select the following 14 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.

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

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

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

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

- *blockTightness* (6): Equation (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.

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

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

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

- *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*.

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

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

- 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 $s \in \mathbf{A}$.

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

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

$$\mathit{shiftBlockTightness} = \{ \mathit{up}_s - \mathit{low}_s \mid s \in \mathbf{A} \} \tag{11}$$

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

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

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

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

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

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

6 Instance Space Analysis

We perform the instance space analysis using the Matlab toolkit MATILDA[2]. 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 and high correlation are removed 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 slight variation in the results.

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*.

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 model `CHUFFED1` as well as the metaheuristic.

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.

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

[2]https://github.com/andremun/matilda

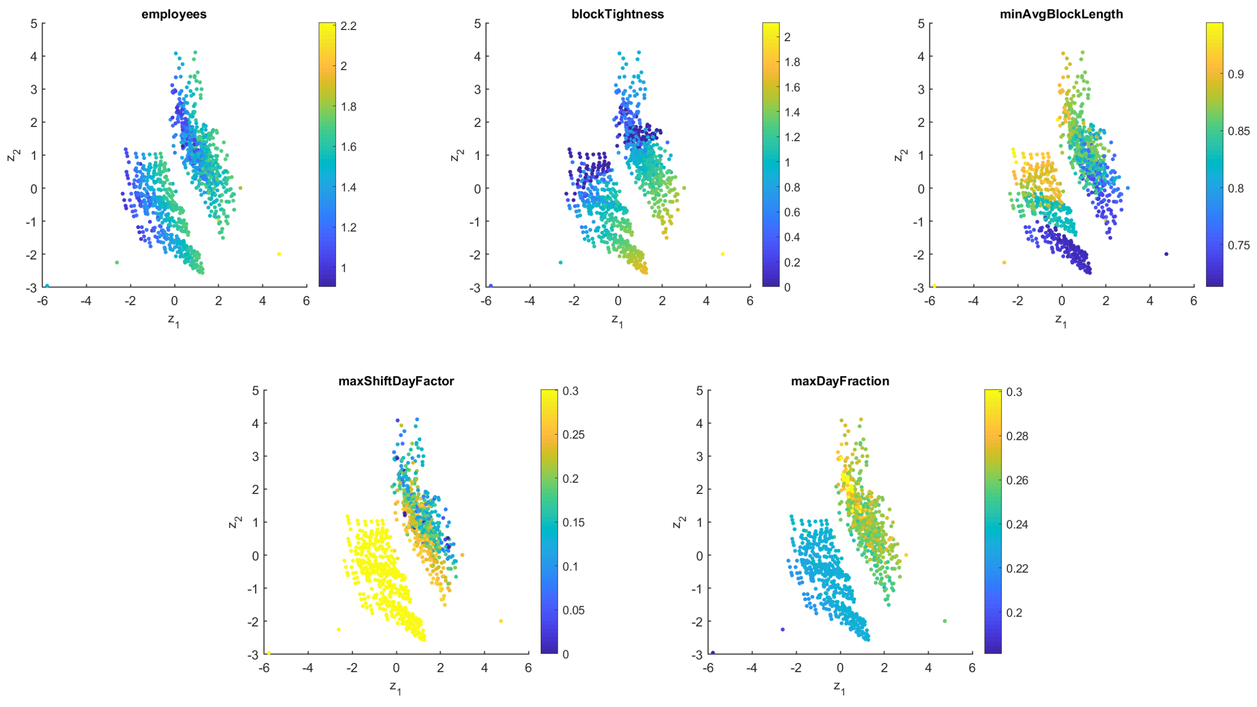


Figure 2: Selected features for the original instance set

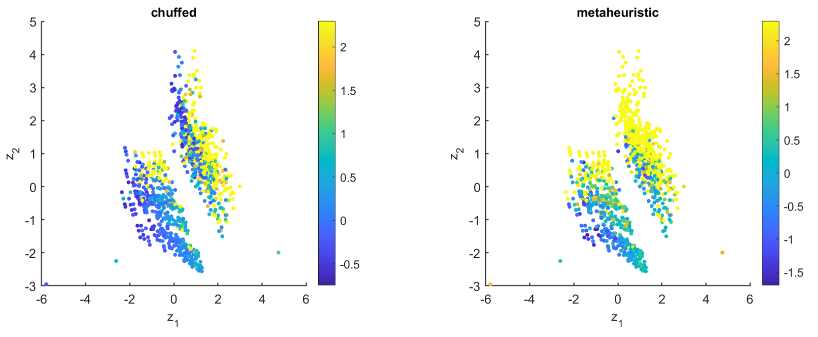


Figure 3: Algorithm results for the original instance set

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 ($maxDayFraction$).

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.

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

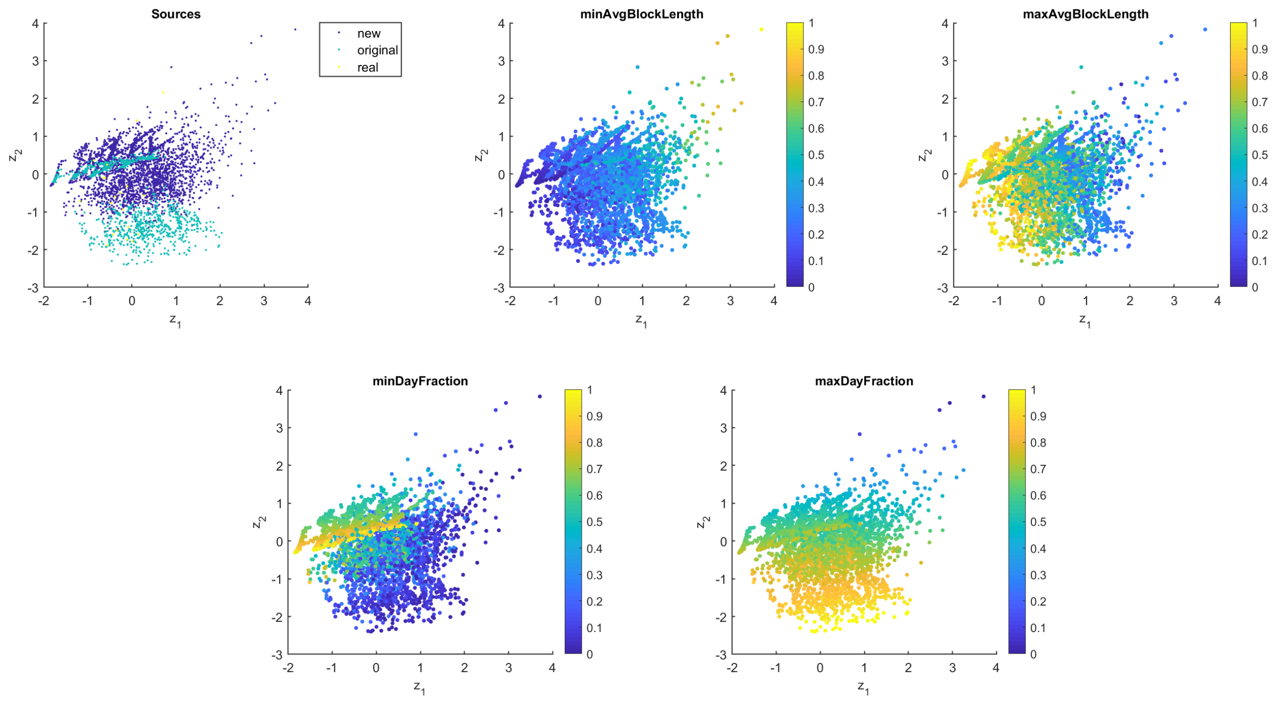


Figure 4: Instance distribution and selected features for the extended instance set

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.

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.

Instance Distribution

Figure 4 shows the new instance space for the extended instance set. 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 projection matrix is given in Equation (16).

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

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 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 Figure 5, therefore, we saw no need to create further instances in this region.

The source chart shows the distribution of original and new instances in the new instance space. The new instances mostly fill the gap between the previous clusters and extends 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.

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 $dayFraction$.

Algorithm Evaluation

Figure 5 shows the algorithm results, again for a timeout of 1000 seconds, this time in the categories feasible solution (blue), proven infeasible (green) 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

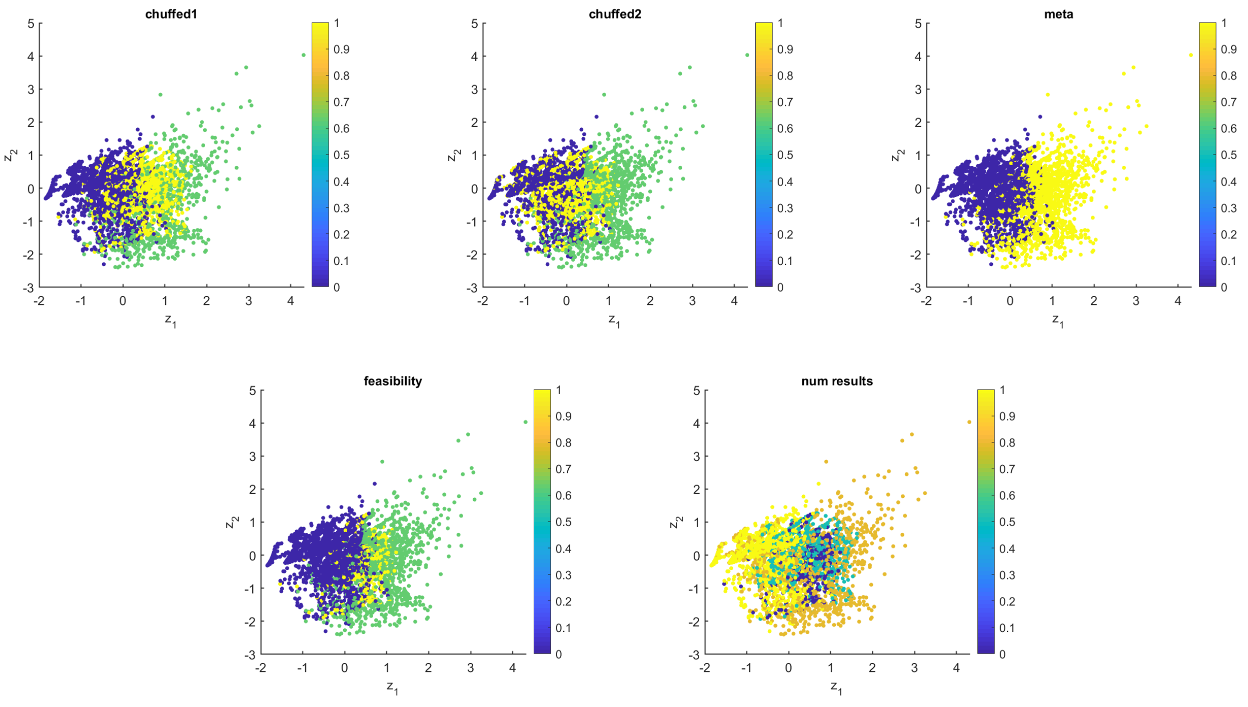


Figure 5: Algorithm results for the extended instance set

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.

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 feasibility 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 (blue), at least one algorithm proved infeasibility (green), or timeout on all algorithms (yellow). First, 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.

The next chart shows the number of solutions (feasible or infeasible) before timeout counted across all algorithms (scale normalized). In the low z_1 region, for most instances all algorithms could find a solution (yellow). For high z_1 or low z_2, both Chuffed algorithms could prove infeasibility for most instances (orange). 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. Furthermore, our set of instances covers both the hard border region and easier regions on either side, indicating a good diversity of instances.

7 Conclusion

We used instance space analysis to evaluate the performance of two constraint models and a metaheuristic on the RWS problem. With the results of the evaluation using the original dataset, we were able to produce a larger, more diverse dataset to cover a larger part of the instance space.

Using the extended dataset, we identified four features that provide a good projection for the analysis of the instance space. We found different strong and weak areas for the individual algorithms and showed that the portfolio of algorithms covers 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.

In future work we could extend the analysis to further dimensions of the problems like larger numbers of shifts or to extended versions of the problem like optimization variants.

Acknowledgments

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

[Baker, 1976] Kenneth R Baker. Workforce allocation in cyclical scheduling problems: A survey. *Journal of the Operational Research Society*, 27(1):155–167, 1976.

[Balakrishnan and Wong, 1990] Nagraj Balakrishnan and Richard T Wong. A network model for the rotating workforce scheduling problem. *Networks*, 20(1):25–42, 1990.

[Chuin Lau, 1996] Hoong Chuin Lau. On the complexity of manpower shift scheduling. *Computers & operations research*, 23(1):93–102, 1996.

[Erkinger and Musliu, 2017] Christoph Erkinger and Nysret Musliu. Personnel scheduling as satisfiability modulo theories. In *International Joint Conference on Artificial Intelligence – IJCAI 2017, Melbourne, Australia, August 19-25, 2017*, pages 614–621, 2017.

[Falcón *et al.*, 2016] Raúl Falcón, Eva Barrena, David Canca, and Gilbert Laporte. Counting and enumerating feasible rotating schedules by means of Gröbner bases. *Mathematics and Computers in Simulation*, 125:139–151, 2016.

[Kang *et al.*, 2017] Y. Kang, R.J. Hyndman, and K. Smith-Miles. Visualising forecasting algorithm performance using time series instance spaces. *Int. J. Forecast*, 33(2):345–358, 2017.

[Kletzander *et al.*, 2019] Lucas Kletzander, Nysret Musliu, Johannes Gärtner, Thomas Krennwallner, and Werner Schafhauser. Exact methods for extended rotating workforce scheduling problems. In *Proceedings of the Twenty-Ninth International Conference on Automated Planning and Scheduling*. American Association for Artificial Intelligence (AAAI), 2019. to appear.

[Laporte and Pesant, 2004] Gilbert Laporte and Gilles Pesant. A general multi-shift scheduling system. *Journal of the Operational Research Society*, 55(11):1208–1217, 2004.

[Laporte *et al.*, 1980] Gilbert Laporte, Yves Nobert, and Jean Biron. Rotating schedules. *European Journal of Operational Research*, 4(1):24–30, 1980.

[Laporte, 1999] G Laporte. The art and science of designing rotating schedules. *Journal of the Operational Research Society*, 50:1011–1017, 9 1999.

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

[Muñoz *et al.*, 2018] Mario A Muñoz, Laura Villanova, Davaatseren Baatar, and Kate Smith-Miles. Instance spaces for machine learning classification. *Machine Learning*, 107(1):109–147, 2018.

[Musliu *et al.*, 2002] Nysret Musliu, Johannes Gärtner, and Wolfgang Slany. Efficient generation of rotating workforce schedules. *Discrete Applied Mathematics*, 118(1-2):85–98, 2002.

[Musliu *et al.*, 2018] Nysret Musliu, Andreas Schutt, and Peter J Stuckey. Solver independent rotating workforce scheduling. In *International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, pages 429–445. Springer, 2018.

[Musliu, 2005] Nysret Musliu. 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*, pages 1529–1530, 2005.

[Musliu, 2006] Nysret Musliu. Heuristic methods for automatic rotating workforce scheduling. *International Journal of Computational Intelligence Research*, 2(4):309–326, 2006.

[Oliveira *et al.*, 2018] Carlos Oliveira, Aldeida Aleti, Lars Grunske, and Kate Smith-Miles. Mapping the effectiveness of automated test suite generation techniques. *IEEE Transactions on Reliability*, 67(3):771–785, 2018.

[Restrepo *et al.*, 2016] María I Restrepo, Bernard Gendron, and Louis-Martin Rousseau. Branch-and-price for personalized multiactivity tour scheduling. *INFORMS Journal on Computing*, 28(2):334–350, 2016.

[Rice, 1976] J.R. Rice. The algorithm selection problem. In *Advances in Computers*, volume 15, pages 65–118. Elsevier, 1976.

[Smith-Miles and Bowly, 2015] K. Smith-Miles and S. Bowly. Generating new test instances by evolving in instance space. *Comput. Oper. Res.*, 63:102–113, 2015.

[Smith-Miles and Lopes, 2012] Kate Smith-Miles and Leo Lopes. Measuring instance difficulty for combinatorial optimization problems. *Computers & Operations Research*, 39(5):875–889, 2012.

[Smith-Miles *et al.*, 2014] K. Smith-Miles, D. Baatar, B. Wreford, and R. Lewis. Towards objective measures of algorithm performance across instance space. *Comput. Oper. Res.*, 45:12–24, 2014.

[Smith-Miles, 2009] Kate A Smith-Miles. Cross-disciplinary perspectives on meta-learning for algorithm selection. *ACM Computing Surveys (CSUR)*, 41(1):6, 2009.

[Triska and Musliu, 2011] Markus Triska and Nysret Musliu. A constraint programming application for rotating workforce scheduling. In *Developing Concepts in Applied Intelligence*, volume 363 of *Studies in Computational Intelligence*, pages 83–88. Springer Berlin / Heidelberg, 2011.