Instance Space Analysis for the Bus Driver Scheduling Problem

Tommaso Mannelli Mazzoli[1][0000-0002-5861-3155], Lucas Kletzander[2][0000-0002-2100-7733], Nysret Musliu[2][0000-0002-3992-8637], and Kate Smith-Miles[3][0000-0003-2718-7680]

[1] TU Wien, Vienna, Austria
[2] Christian Doppler Laboratory for Artificial Intelligence and Optimization for Planning and Scheduling, DBAI, TU Wien, Vienna, Austria
{tommaso.mazzoli,lucas.kletzander,nysret.musliu}@tuwien.ac.at
[3] ARC Training Centre in Optimisation Technologies, Integrated Methodologies and Applications (OPTIMA), School of Mathematics and Statistics, The University of Melbourne, Parkville, Victoria, Australia smith-miles@unimelb.edu.au

Abstract. The Bus Driver Scheduling Problem is a combinatorial optimisation problem with important real-world applications. The goal is to assign bus drivers to predetermined bus tours in order to minimise an objective function that considers the total time of each employee's one-day work shift and their dissatisfaction.

Due to the amount of complex rules specified by a collective agreement and European laws, this problem is highly constrained. Thus, exact methods are computationally intractable. In recent work, two metaheuristics have been proposed to solve this problem: Large Neighbourhood Search (LNS) and Construct, Merge, Solve and Adapt (CMSA). In the literature, 65 real-world-like instances have been used to test the algorithms. Among those instances, LNS seems to outperform CMSA; nevertheless, the reason was still obscure.

In order to investigate the reason, we use Instance Space Analysis to show the weaknesses and strengths of the two solution methods. First, we greatly extend an instance generator to be able to generate varied real-world-like and synthetic instances. This allows us to expand the previous set of instances from the literature.

We then present a set of features that describe the hardness of the instances. The features consider the structure of the instance, such as the average break length for each vehicle or the distribution of bus tours in the city. We observe that even if LNS outperforms CMSA in real-world-like instances, it does not for some synthetic ones.

Using Instance Space Analysis, each instance is projected into a 2D plane based on selected features. We see clusters of instances in the instance space, and the real-world-like are in the centre. The bus tour structure appears to have an impact on the performance of the algorithms. Using this information, we can gain insights into the weaknesses and strengths of the two algorithms.

Keywords: Instance Space Analysis, Bus Driver Scheduling Problem, Scheduling, Combinatorial Optimisation

1 Introduction

This work deals with the Bus Driver Scheduling Problem and how to (1) generate new *diverse* instances and (2) how to objectively assess how the state-of-the-art algorithms perform on those diverse instances.

The Bus Driver Scheduling Problem (BDSP) is a sub-problem of the general Transportation Planning System, that includes Vehicle Scheduling, Crew Rostering, and Timetabling [11]. The goal is to assign bus drivers to predetermined routes while minimising a specified objective function that considers operating costs as well as employee dissatisfaction with their work shifts.

The problem has a clear practical relevance. Recently, a variant of the BDSP with complex break constraints has been studied [2,3,1,7,5]. In this setting, there a set of 65 real-world-like instances (named *Realistic*) and performance results for two state-of-the-art algorithms: CMSA [7] and LNS [5]. Based on the 65 Realistic instances, LNS appears to outperform CMSA. However, we must scrutinise the diversity of these test instances and seek to ensure they span a range of instance characteristics before conclusions can be drawn about the merits of each algorithm.

Instance Space Analysis [10] is a methodology that allows the diversity of a set of test instances to be visually examined, and insights into the strengths and weaknesses of algorithms, across the mathematically defined boundaries of the instance space, to be observed.

Our research contributions of this paper are:

- We developed an instance generator with whom we have generated diverse instances.
- We propose a set of features of the test instances to characterise their similarities and differences.
- We compare the two main state-of-the-art algorithms for the BDSP on real-world-like and synthetic instances.
- We scrutinise the instance space and point out the regions of instance space that are not yet sufficiently covered.

This paper is organised as follows: Section 2 describes the Bus Driver Scheduling Problem in detail. In Section 3 we present the Instance Space Analysis framework, and the meta-data we built for the Instance Space Analysis. In Section 4, we show and visualise the Instance Space Analysis. Finally, in Section 5 we present the conclusions of the work and its future possible developments.

2 The Bus Driver Scheduling Problem

The investigated Bus Driver Scheduling Problem deals with the assignment of bus drivers to vehicles that already have a predetermined route for one day of operation. The problem specification is taken from the literature [2].

2.1 Problem Input

The input of the BDSP consists in three pieces of data:

- For a set P of *positions* (for instance, a set of bus stops), a time distance matrix $D = (d_{ij}) \in \mathbb{R}^{|P|\times|P|}$ is given, where d_{ij} represents the time needed for an employee to go from position i to j when not actively driving a bus. If no transfer is possible, then $d_{ij} = M$, where M is a very large number. If $i \neq j$, then d_{ij} is called *passive ride time*, whereas d_{ii} represents the time it takes to switch vehicle at the same position, but is not considered passive ride time.
- For each position $p \in P$ two values $startWork_p$ and $endWork_p$: they represent respectively the amount of working time required to start or end a work shift at that position.
- A set L of *bus legs*: each leg $\ell \in L$ is a 5-tuple:

$$\ell = (tour_\ell, startPos_\ell, endPos_\ell, start_\ell, end_\ell),$$

  representing the trip of a vehicle between two stops at a certain time:
  - $tour_\ell \in \mathbb{N}$ is the ID of the vehicle
  - $startPos_\ell, endPos_\ell \in P$ are respectively the starting and the ending positions of the leg
  - $start_\ell \in \mathbb{R}$ is the time when the vehicle departs from position $startPos_\ell \in \mathbb{R}$
  - $end_\ell \in \mathbb{R}$ is the time at which the vehicle arrives to position $endPos_\ell$

  Legs with the same tour t do not overlap: the intervals $(start_\ell, end_\ell)$ for ℓ such that $tour_\ell = t$ are disjoint. The set L is totally ordered by *start*, using *tour* as tie-breaker.

ℓ	$tour_\ell$	$start_\ell$	end_ℓ	$startPos_\ell$	$endPos_\ell$
1	1	420	495	0	1
2	1	520	530	1	2
3	1	540	550	2	1
4	1	558	570	1	0

Table 1: A Bus Tour Example

Table 1 shows a short example of one particular bus tour. The vehicle starts at time 420 (6:40 AM) at position 0, does multiple legs between positions 1 and 2 with waiting times in between and finally returns to position 0.

For Realistic instances, the number of legs is proportional to the number of bus tours with approximately $n_{\text{legs}} \approx 10 \cdot n_{\text{tours}}$.

2.2 Solution

A solution S to the problem is an assignment $S\colon L \to E$, where $E \subseteq \mathbb{N}$ is the set of employees. The number of drivers is not given, but one can imagine setting it as large as needed to have a feasible solution.

Equivalently, it is useful to represent a solution by a set of *shifts*, that is the work scheduled to be performed by a driver in one day [11]. More precisely, the shift of driver

$e \in E$ is the preimage $L_e = S^{-1}(\{e\})$ with the total order induced by L. Hence, the notion of *consecutive* bus legs in a shift is well-defined.

Each shift of a driver $e \in E$ must be feasible according to the following criteria:

- No overlapping bus legs are assigned to e.
- Changing tour or position between consecutive legs $i, j \in L_e$ requires

$$start_j \geq end_i + d_{endPos_i, startPos_j}.$$

- The shift L_e respects all hard constraints regarding work regulations as specified below. These refer to different measures of time related to an employee e containing the set of bus legs L_e, as visualised in Figure 1.

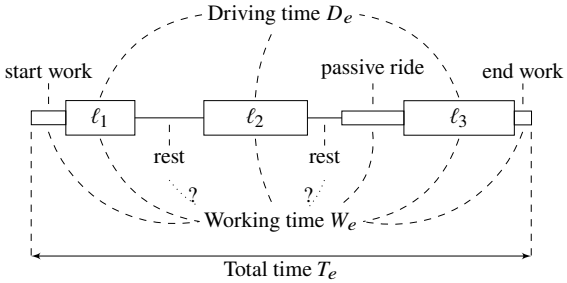


Fig. 1: Example shift [2]

Driving Time Regulations. The *driving time* of a shift L_e is

$$D_e = \sum_{i \in L_e} (end_i - start_i)$$

The driving time D_e cannot exceed $D_{\max} = 9\,\text{h}$. The driving time is subject to additional rules regarding driving breaks. A *driving break* between two consecutive bus legs i and j is

$$diff_{ij} = start_j - end_i$$

After at most 4 h of driving time, one of the following has to occur:

- One driving break of at least 30 min.
- Two driving breaks of at least 20 min each.
- Three driving breaks of at least 15 min each.

Once we reach all required breaks, the next block of at most 4 h starts.

Total Time Regulations. The *total time* of a shift L_e is the span from the start of work until the end of work:

$$T_e = end_\ell + endWork_{endPos_\ell} - (start_f - startWork_{startPos_f})$$

where f is the first leg and ℓ is the last leg in the shift L_e. A hard limit of $T_e \leq T_{\max} = 14\,\text{h}$ is enforced.

Shift Splits. We say that the employee e has a *shift split* if L_e contains two consecutive legs i and j such that:

$$start_j - end_i - r_{endPos_i, startPos_j} \geq 3\,\text{h}$$

where $r_{p,q} = d_{p,q}$ if $p \neq q$, else $r_{p,p} = 0$. Denote by $split_e$ the number of shift splits and by $splitTime_e$ the total amount of time the driver e spends on a shift split. A shift split resets the driving time (i.e., it counts as a *driving* break). A shift contains up to two shift splits.

Shift splits are unpaid, so they are badly regarded by bus drivers. This will play a role in designing the objective function.

Working Time Regulations. The working time W_e cannot exceed 10 h and has a soft minimum of 6.5 h. If the employee is working for a shorter period of time, the difference has to be paid anyway.

A minimum rest break is required according to the following options:

- $W_e < 6\,\text{h}$: no rest break;
- $6\,\text{h} \leq W_e \leq 9\,\text{h}$: at least 30 min;
- $W_e > 9\,\text{h}$: at least 45 min.

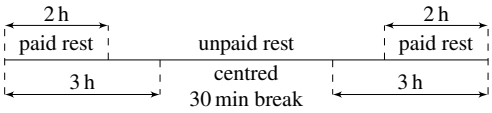


Fig. 2: Rest break positioning [2]

The rest break may be split into one part of at least 30 min and one or more parts of at least 15 min. The first part has to occur after at most 6 h of working time. Whether rest breaks are paid or unpaid depends on break positions according to Figure 2. Every period of at least 15 min of consecutive rest break is unpaid as long as it does not intersect the first 2 or the last 2 hours of the shift (a longer rest break might be partially paid and partially unpaid). The maximum amount of unpaid rest is limited:

- If 30 consecutive minutes of rest break are located such that they do not intersect the first 3 h of the shift or the last 3 h of the shift, at most 1.5 h of unpaid rest are allowed;
- Otherwise, at most one hour of unpaid rest is allowed.

2.3 Objective function

We minimise the objective function combining cost and employee satisfaction defined in previous work [2]:

$$z = \sum_{e \in E} \left(2\, W'_e + T_e + ride_e + 30\, change_e + 180\, split_e\right) \tag{1}$$

The objective function z represents a linear combination of six criteria for each employee e. The actual paid working time $W'_e = \max\{W_e, 390\}$ is the main objective (containing actual working time and additional payments for short shifts), and it is combined with the total time T_e to reduce long unpaid periods for employees. The next sub-objectives reduce the passive ride time $ride_e$ and the number of tours changes $change_e$, which is beneficial for both employees and efficient schedules. The last objective aims to reduce the number of split shifts $split_e$ as they are very unpopular. The weights were determined by previous work [2] based on preferences agreed by different stakeholders at Austrian bus companies and employee scheduling experts. Details of the objective function can be found in previous work [2,3,1].

3 Instance Space Analysis

Instance Space Analysis (ISA) is a methodology proposed by Smith-Miles et al. in 2014 [8] that extends the algorithm selection problem framework of Rice [6]. In ISA, instances are represented as vectors of features. The instances are then projected onto the 2D plane to separate hard and easy instances. Figure 3 illustrates the Instance Space Analysis framework.

The *problem space* $\mathcal{P}$ contains all the theoretically possible instances of the BDSP. Nevertheless, we only have results for a (smaller) subset of instances $\mathbf{I} \subset \mathcal{P}$, for which we have experiment results.

The first component of the meta-data are some chosen features, used to characterize the mathematical and statistical properties of the instances that (1) describe the similarities and differences between instances in (2) have correlation with the performance of one or more algorithms.

For a given instance $x \in \mathbf{I}$, we calculate the feature vector $\boldsymbol{f}(x)$, which represents an instance in the *feature space*, $\mathcal{F}$.

The second component are the performance measures. We imagine to have the algorithm space $\mathcal{A}$ representing the set of algorithms available to solve all instances in $\mathbf{I}$. For each algorithm $\alpha \in \mathcal{A}$ and each instance $x \in \mathbf{I}$, we have a performance measure, $y(\alpha, x)$: an element of a vector that describes the performance space, $\mathcal{Y}$, and requires a user-defined measure of "goodness," such as the objective function value obtained for a fixed computational budget. Both the features and performance measures for all the instances in $\mathbf{I}$, and all algorithms in $\mathcal{A}$ constitute the meta-data, which we represent as two matrices $\boldsymbol{F} = [f_1, \ldots, f_n] \in \mathbb{R}^{m \times n}$ and $\boldsymbol{Y} = [y_1, \ldots y_n] \in \mathbb{R}^{a \times n}$, where m is the number of features, n is the number of problem instances, and a is the number of algorithms. Hence, the meta-data is the set $\{\boldsymbol{F}, \boldsymbol{Y}\}$.

In the original framework proposed by Rice in 1976 [6], a selection mapping S was learned directly from features and performance. Later, in the expanded framework

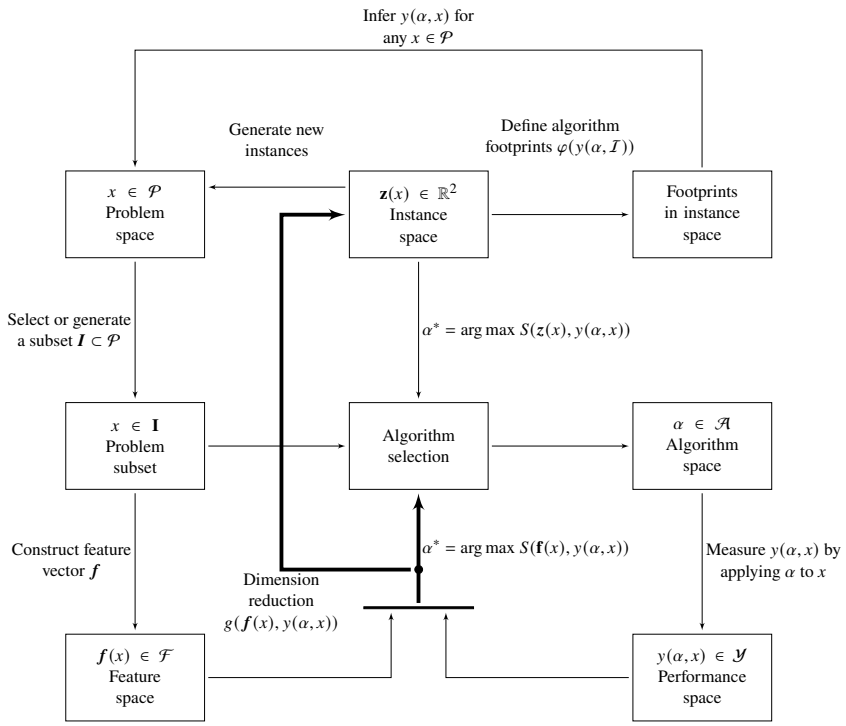


Fig. 3: ISA framework [10] extending the original by Rice [6]

introduced by Smith-Miles et al. in 2014 [8], and extended by Smith-Miles and Muñoz in 2023 [10], instances are projected from the feature space into a lower-dimensional 2D space using the dimension reduction $g\left(\boldsymbol{f}(x), y(\alpha, x)\right)$. It aims to yield an optimal projection that looks for linear trends in both features and algorithmic performance across the resultant instance space, in order to gain interpretable insights. This allows us to get a visualization and enables a more detailed evaluation of algorithmic performance, as well as algorithm selection based on the position of an instance in the instance space.

In the conceptual framework delineated in the preceding section, the ISA methodology involves six fundamental steps:

1. Acquiring experimental meta-data for a designated set of instances $\mathbf{I}$ ran across a set of algorithms $\mathcal{A}$. This includes capturing feature values $\boldsymbol{F}$ for all instances and recording performance metrics $\boldsymbol{Y}$ for all algorithms across all instances.
2. Creating an instance space through a feature selection process applied to the meta-data $\{\boldsymbol{F}, \boldsymbol{Y}\}$, with consideration for a user-defined benchmark for optimal performance, encompassing its theoretical boundaries.
3. Employing machine learning techniques to generate predictions for automated algorithm selection.

4. Establishing algorithm footprints and evaluating associated metrics.
5. Analysing the instance space to extract insights and assess the adequacy of the available meta-data.
6. Generating supplementary meta-data as needed, and iterating from Step 2 onwards, or concluding the process if no further iterations are warranted.

3.1 Problem Subset *I*

In 2020, Kletzander and Musliu [2] proposed a set of 50 real-world-like instances for the BDSP. This set was later [1] extended with 15 new (again real-world-like) instances. The instances are publicly available[4]. For these instances, the number of bus tours ranges from 10 bus tours (about 70 legs) up to 250 (about 2300 bus legs). These instances are build to reproduce particular properties seen in a specific industrial use-case, however, in other settings involving different locations or rule sets, real-world instances might exhibit very different properties.

In order to cover a larger portion of the instance space, we greatly extended the original instance generator. It can generate instances with a larger number of bus stops and more diverse distributions of bus legs and tours during the day. The generator uses 44 parameters.

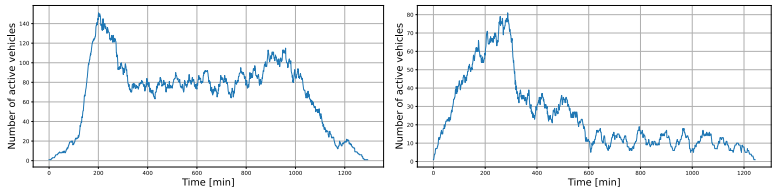
We changed some of this parameters (one at time) and we then generated 219 new diverse instances. The new instances are divided into 12 distinct class families (named sources). A brief description of the types is given in Table 2.

Table 2: The 12 types of instance sources. The third column gives the value for the existing benchmark instances.

Name	Characteristic	Standard
breakMax	No breaks between two consecutive bus legs	$[3, 35]$ min
distanceAvailability	The probability that 2 stops are connected is 0.1	0.9
distanceVariation	Add uniform $\mathcal{U}_{[1,100]}$ distance perturbation	$\mathcal{U}_{[1,1.2]}$
legRegularity	Probability of reusing the last leg is 0.1	0.9
numStations	There are 1000 bus stops	10
morningPeak	Morning peak is 5 times the regular demand	1.8
legPeriodMax	Max number of break lengths in use per tour is 5	3
shortLeg	Every leg length is in the interval $[5, 15]$ min	$[20, 60]$ min
gridSpread	Bus stops drawn using the distribution $\mathcal{N}^2(0, 1000)$	$\mathcal{N}^2(0, 50)$
legMax	The maximum leg length is 240 min	60 min
legMin	The minimum leg length is 5 min	20 min

Figure 4 shows the demand distribution of two instances. In both cases there is a significant morning peak when both employees and students need numerous buses within a brief period, followed by a decrease in activity. With the instance generator, we can create instances like in Figure 4b, where the peak in the morning is extremely high.

[4] https://cdlab-artis.dbai.tuwien.ac.at/papers/sa-bds/



(a) Real-life-like (b) Synthetic

Fig. 4: Demand Distribution of active vehicles.

3.2 Algorithm Space $\mathcal{A}$

We ran Instance Space Analysis with two algorithms from previous work.

The first algorithm is Construct, Merge, Solve and Adapt (CMSA) [7], a matheuristic algorithm.

The second algorithm is Large Neighbourhood Search (LNS) [5], where a destroy operator creates a sub-instance of the BDSP by removing all the legs of a number of drivers based on tours that are shared by these drivers. Then, LNS uses Column Generation as repair operator. Even if the solution of the sub-problem is not optimal, it is in fact close enough to the optimal with a very small optimality gap.

We set 5 min as timeout for both algorithm. We considered the average of the objective function values over 5 runs. All executions were performed on a cluster with 11 nodes using Ubuntu 22.04.2 LTS. Each node has two Intel Xeon E5-2650 v4 (max 2.20 GHz, f12 physical cores, no hyperthreading). For each run, we set a memory limit of 4.267 GB and use one thread. The implementation is in Python, executed with PyPy 7.3.11. Column Generation is implemented in Java, using OpenJDK 20, and CPLEX 22.11 for the master problem.

3.3 Feature Space $\mathcal{F}$

In order to describe the difficulty of each instance, we collect a set of features. A feature is a number that characterizes a proprieties of an instances. We collect a set of 84 features, described in Table 3.

A special feature is the number of relative relief opportunities of an instance, defined as follow.

Definition 1 (Relative Relief Opportunity (RRO)). *Let $p \in P$ be a position and $t \in \mathbb{R}$. We define a relief opportunity in p at time t (in minutes) as the proportion of bus legs that are passing through position p in the time window $[t, t + 60\,\text{min})$:*

$$RRO(p,t) = \frac{|\{\ell \in L \mid startPos_\ell = p \ \wedge \ start \in [t, t+60)\}|}{|\{\ell \in L \mid startPos_\ell = p\}|}$$

Note that for each position $p \in P$, we can evaluate, e.g., $\max_t RRO(p,t)$ and then $\max_{p \in P} \max_t RRO(p,t)$ that tells us what is the maximum relief opportunity across the positions throughout the day.

Table 3: Set of 84 features used in Meta-Data. With *glorious seven* we mean the seven descriptive statistics: Max, Min, Average, Median, Std, First quartile and Third quartile.

Feature Name	Description
Size-related: Dimension of the problem (4 features)	
Number of Tours	Number of distinct bus tours of the problem
Number of Legs	Number of distinct bus legs of the problem
Number of Positions	Number of bus stops (positions) used
Number of Active vehicles	Max number of active vehicles during the day
Geometry: (1 feature)	
Average distance	Average distance between bus stops
Bus Tours: Glorious seven across all tours for each feature (35 features)	
Total Time per tour	Total span time for each tour
Number of breaks per tour	Number of breaks between consecutive legs
Number of proper breaks per tour	Number of breaks of ≥ 15 min for each tour
Number of legs per tour	Number of bus legs for each tour
Number of large legs per tour	Number of legs with length ≥ 2 h for each tour
Distributions: Glorious seven across all legs for each feature (14 features)	
Drive	Bus legs lengths
Breaks statistics	Length of breaks between consecutive legs
RRO: Max, Min, Avg, Median, Std across all positions (25 features)	
Max RRO	Max Number of RROs over the time horizon
Min RRO	Min Number of RROs over the time horizon
Mean RRO	Mean Number of RROs over the time horizon
Median RRO	Median Number of RROs over the time horizon
Std RRO	Std Number of RROs over the time horizon
Bin Packing Problem [4]**:** k is the longest leg length (5 features)	
Huge	Proportion of legs that have length $\lvert\ell\rvert > k/2$
Large	Proportion of legs with $k/3 < \lvert\ell\rvert \leq k/2$
Medium	Proportion of legs with $k/4 < \lvert\ell\rvert \leq k/3$
Small	Proportion of legs with $k/10 < \lvert\ell\rvert \leq k/4$
Tiny	Proportion of legs with $\lvert\ell\rvert \leq k/10$

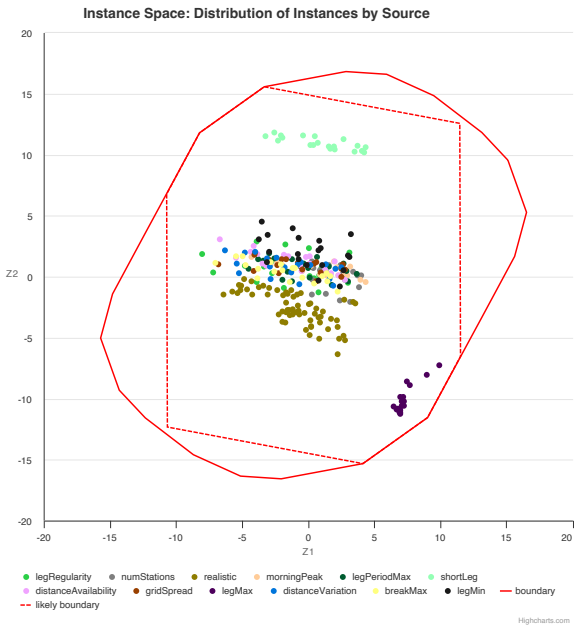


Fig. 5: Bus Driver Scheduling Problem instance space defined by Equation (2). We recognise three main clusters: legMax, shortLeg and all the rest. We also observe that the Realistic instances (in brown) are in the middle of the Instance Space.

4 Results and Evaluation

We perform the Instance Space Analysis using the Matlab toolkit MATILDA available from https://matilda.unimelb.edu.au. The settings for MATILDA are the default settings except the performance threshold set as 0.0.

Summing up, we have 284 instances from two distinct sources, 84 features described in Table 3, and 2 algorithms: CMSA [7] and LNS [5].

4.1 Instance Distribution

Figure 5 shows the distribution of the instance sources across the instance space. We notice that Realistic instances are located around the centre of the instance space, meaning that the feature values are average. Moreover, shortLeg and legMax instances appear to be close to the theoretical boundary of the Instance Space. Those are instances where the leg length has drastically changed from the value of the Realistic instances.

The red solid outer line represents the theoretical boundaries made by considering all the feasible combinations of features and their upper and lower bounds. The red dotted line instead is the likely boundary.

In total, we have 84 features. Equation (2) shows the projection matrix applied to the ten features after the preprocessing (that includes normalisation and Box-Cox transformation). We observe that four out of ten features describe the distribution of *drive*, that is, the leg length. Two features are related to the Total Time per Tour, i.e., the maximum and minimum total span of each tour, from the very beginning to the very end. The other four features represent the distribution of the Relative Relief Opportunities.

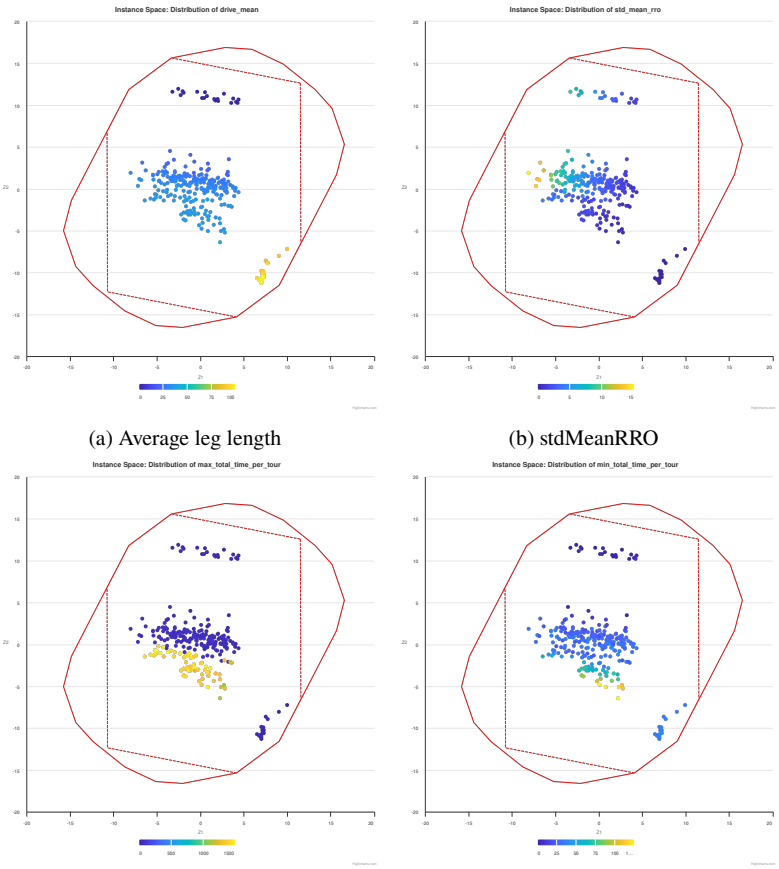
$$\begin{bmatrix} Z_1 \\ Z_2 \end{bmatrix} = \begin{bmatrix} 0.2636 & -0.7982 \\ -0.1663 & -0.6664 \\ 0.6380 & -0.5291 \\ -0.7371 & -1.4743 \\ -0.6819 & -0.5257 \\ -0.4333 & -0.8723 \\ -0.7684 & 0.3775 \\ -1.3987 & -0.2818 \\ -0.4252 & 0.4900 \\ -0.6950 & 0.0500 \end{bmatrix}^{\mathrm{T}} \cdot \begin{bmatrix} \mathit{driveMax} \\ \mathit{driveMean} \\ \mathit{driveMedian} \\ \mathit{drive3rdQuartile} \\ \mathit{maxTotalTimePerTour} \\ \mathit{minTotalTimePerTour} \\ \mathit{stdMaxRro} \\ \mathit{stdMeanRro} \\ \mathit{stdMedianRro} \\ \mathit{stdStdRro} \end{bmatrix} \tag{2}$$

Figure 6 shows the distribution of four features. In Figure 6a we see that, as expected, the average length of bus legs is very high for legMax instances (where the leg length belongs to the interval $[20, 240]$), whereas for shortLeg instances is extremely low (here the leg length belongs to the interval $[5, 15]$). Figure 6b shows the distribution of the standard deviation (over the bus stops) of the minimum number of Relative Relief Opportunities during the day. This essentially is related to the number of possible changes/moves that we can do during the day for each bus stop. This value appears to be very low among the legMax instances, where the legs are usually large and, therefore the number of possible vehicle changes during the day is reduced. The other two images are about the total time per tour. In Figure 6c we see a clear distinction between Realistic instances and the new generated one. This is because the new generated instances have a lower maximum of length of bus tours. Figure 6d shows the Minimum total time per tour.

4.2 Algorithm Evaluation

Fig. 7 shows the binary performance distribution of the two algorithms. We observe that in the middle cluster, LNS performs better than CMSA. However, CMSA appears to have better results closer to the theoretical boundaries. Fig. 8 shows the prediction of the Support Vector Machine, supporting this idea.

We believe that the "structure" of the bus tour impacts the performance of the two algorithms. In particular, LNS removes all the bus legs with some selected bus tours. In contrast, CMSA randomly generates a number of greedy solutions at every iteration and, therefore, does not directly exploit the bus tours. LNS seems to perform better than CMSA for most of the instances (including Realistic), but not for all. We observe that CMSA gets better solutions for legMax and shortLeg instances. These instances have very short tours. Thus, LNS does not benefit much from removing all the legs associated with the same tour. Hence, CMSA (which does not explicitly depend on the structure of bus tours) provides good results with no significant difference from the others.



(a) Average leg length

(b) stdMeanRRO

(c) Max Total Time x Tour

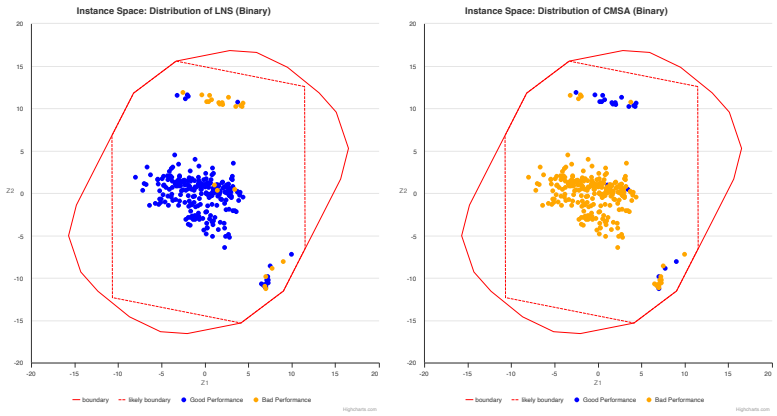
(d) Min Total Time x tour

Fig. 6: Distribution of features. The colors represent the feature values. Axes as defined by the equation (2).

4.3 Filling the Gaps

Thanks to Instance Space Analysis, we observe that there is still a considerably extended region between the four clusters in the instance space. This reveals opportunities to generate new instances in order to fill this gap. A first possible way to do that is by changing or adding parameters of the instance generator and trying to explore the instance space.

A more elaborated one is to fix a target (e.g., a portion of the Instance Space to fill up) and generate instances through a Genetic Algorithm that evolves new instances in



(a) LNS (b) CMSA

Fig. 7: Binary performance distribution. We see that CMSA performs better in some of the new generated instances close to the boundary.

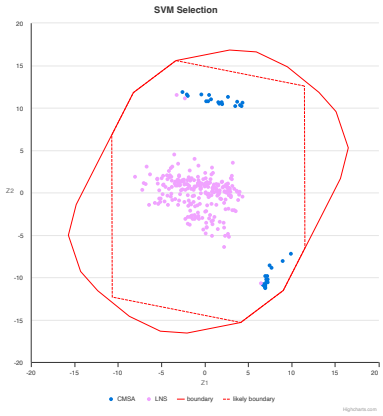


Fig. 8: Algorithm selection using SVM

the desirable region, as done by Smith-Miles [9]. However, this procedure is problem-dependent and requires more investigation.

5 Conclusions and Future Investigations

In this paper, we have applied Instance Space Analysis to the Bus Driver Scheduling Problem for the first time. We evaluated the performance of two metaheuristic techniques for the BDSP, providing insights into the strength of LNS and the boundaries of its good performance. We greatly increased the capabilities of the instance generator and extended the previous set of instances with new, diverse ones. We defined and evaluated a novel set of features, seeing which features help the most to explain algorithm performance.

In the future, we want to fill the instance space by creating more instances that are even more diverse than the ones present now. At first, we will consider other public transportation systems, possibly located in different countries. Then, we will create instances with new combinations of parameters like long leg lengths and short bus tour lengths. Ideally, we want to use a Genetic Algorithm to automatically evolve the instances to fill up certain regions in the Instance Space. The goal is to perform automatic algorithm selection and outline the region of the instance space where one algorithm performs better than another. Thanks to this problem's structure, this will also be helpful for related problems such as vehicle routing.

Furthermore, we will test other solution methods using other quality metrics, such as the GAP from the best-known solution or the area under the curve of the trajectory of solutions found during the search.

Acknowledgements. The financial support by the Austrian Federal Ministry of Labour and Economy, the National Foundation for Research, Technology and Development and the Christian Doppler Research. This work is also partially funded by the Doctoral Program Vienna Graduate School on Computational Optimization, Austrian Science Foundation (FWF), under grant No.: W1260-N35 (https://vgsco.univie.ac.at/).

References

1. Kletzander, L., Mazzoli, T.M., Musliu, N.: Metaheuristic algorithms for the bus driver scheduling problem with complex break constraints. In: Proceedings of the Genetic and Evolutionary Computation Conference. p. 232–240. GECCO '22, Association for Computing Machinery, New York, NY, USA (2022). https://doi.org/10.1145/3512290.3528876
2. Kletzander, L., Musliu, N.: Solving large real-life bus driver scheduling problems with complex break constraints. Proceedings of the International Conference on Automated Planning and Scheduling **30**(1), 421–429 (Jun 2020), https://ojs.aaai.org/index.php/ICAPS/article/view/6688
3. Kletzander, L., Musliu, N., Van Hentenryck, P.: Branch and price for bus driver scheduling with complex break constraints. Proceedings of the AAAI Conference on Artificial Intelligence **35**(13), 11853–11861 (May 2021), https://ojs.aaai.org/index.php/AAAI/article/view/17408
4. Liu, K., Smith-Miles, K., Costa, A.: Using Instance Space Analysis to Study the Bin Packing Problem. Ph.D. thesis, PhD thesis (2020)

5. Mazzoli, T.M., Kletzander, L., Hentenryck, P.V., Musliu, N.: Investigating large neighbourhood search for bus driver scheduling. In: 34th International Conference on Automated Planning and Scheduling (2024), https://openreview.net/forum?id=d4TzG4ivNu
6. Rice, J.R.: The algorithm selection problem. In: Rubinoff, M., Yovits, M.C. (eds.) Advances in Computers, Advances in Computers, vol. 15, pp. 65–118. Elsevier (1976). https://doi.org/https://doi.org/10.1016/S0065-2458(08)60520-3
7. Rosati, R.M., Kletzander, L., Blum, C., Musliu, N., Schaerf, A.: Construct, merge, solve and adapt applied to a bus driver scheduling problem with complex break constraints. In: AIxIA 2022 – Advances in Artificial Intelligence, pp. 254–267. Springer International Publishing (2023). https://doi.org/10.1007/978-3-031-27181-6_18
8. Smith-Miles, K., Baatar, D., Wreford, B., Lewis, R.: Towards objective measures of algorithm performance across instance space. Computers& Operations Research **45**, 12–24 (2014). https://doi.org/https://doi.org/10.1016/j.cor.2013.11.015, https://www.sciencedirect.com/science/article/pii/S0305054813003389
9. Smith-Miles, K., Bowly, S.: Generating new test instances by evolving in instance space. Computers & Operations Research **63**, 102–113 (2015)
10. Smith-Miles, K., Muñoz, M.A.: Instance space analysis for algorithm testing: Methodology and software tools. ACM Comput. Surv. **55**(12) (mar 2023). https://doi.org/10.1145/3572895
11. Wren, A.: Scheduling vehicles and their drivers-forty years' experience. Tech. rep., University of Leed (2004)