The maximum length car sequencing problem

Lara di Cavalcanti Pontes

CENTRO DE INFORMÁTICA
UNIVERSIDADE FEDERAL DA PARAÍBA

João Pessoa
2024

Lara di Cavalcanti Pontes

The maximum length car sequencing problem

Monograph presented in fulfillment of the requirements for the Bachelor's degree in Computer Engineering, Centro de Informática, Universidade Federal da Paraíba.

Advisor: Anand Subramanian

João Pessoa
2024

Catalogação na publicação
Seção de Catalogação e Classificação

P814m Pontes, Lara di Cavalcanti.
The maximum length car sequencing problem / Lara di Cavalcanti Pontes. - João Pessoa, 2024.
85 f. : il.

Orientação: Anand Subramanian.
TCC (Graduação) - UFPB/Informática.

1. Scheduling. 2. Assembly line. 3. Car sequencing. 4. Combinatorial optimization. I. Subramanian, Anand. II. Título.

UFPB/CI CDU 004:629

Elaborado por Michelle de Kássia Fonseca Barbosa - CRB-738

CENTRO DE INFORMÁTICA
UNIVERSIDADE FEDERAL DA PARAÍBA

Work titled ***The maximum length car sequencing problem*** presented in fulfillment of the requirements for the Bachelor's degree in Computer Engineering, approved by:

Documento assinado digitalmente
gov.br
ANAND SUBRAMANIAN
Data: 22/05/2024 12:07:48-0300
Verifique em https://validar.iti.gov.br

Prof. Dr. Anand Subramanian
Universidade Federal da Paraíba

Documento assinado digitalmente
gov.br
TEOBALDO LEITE BULHOES JUNIOR
Data: 22/05/2024 10:04:28-0300
Verifique em https://validar.iti.gov.br

Prof. Dr. Teobaldo Leite Bulhões Júnior
Universidade Federal da Paraíba

Documento assinado digitalmente
gov.br
LUCIANO CARLOS AZEVEDO DA COSTA
Data: 22/05/2024 09:56:42-0300
Verifique em https://validar.iti.gov.br

Prof. Dr. Luciano Carlos Azevedo da Costa
Universidade Federal da Paraíba

João Pessoa, May 13, 2024

Centro de Informática, Universidade Federal da Paraíba
Rua dos Escoteiros, Mangabeira VII, João Pessoa, Paraíba, Brasil CEP: 58058-600
Fone: +55 (83) 3216 7093 / Fax: +55 (83) 3216 7117

To my parents,
who always ensured a feasible solution space for me to pursue my objective function.

Acknowledgments

I would like to express my deepest gratitude to my advisor, Prof. Anand Subramanian, for first presenting me to the world of operations research and for his continuous and attentive guidance throughout my undergraduate years. Also, I am extremely grateful to my co-advisor, Prof. Maria Battarra, for the invaluable lessons and for warmly welcoming me during my exchange period at the University of Bath. And special thanks to my brilliant colleague Carlos Neves for his collaboration in this work. This endeavor would not have been possible without them.

I would like to extend my sincere thanks to Prof. Teobaldo Bulhões and Prof. Luciano Costa for composing the examining committee and for the helpful comments, Prof. Bruno Bruck and Prof. Thaís Gaudencio for their support during my academic pursuits, Prof. Artur Pessoa for his valuable suggestions regarding the mathematical formulations, Dr. Yuan Sun for kindly providing the set of instances used in his study, and the anonymous reviewers for the relevant insights that considerably improved the quality of the work.

I cannot express enough gratitude to my parents, Soraia Pinheiro and Carlos Pontes, for being my core foundation and biggest supporters. On the same note, I profoundly thank Louise Pinheiro, Lívia Pinheiro, Marina Pontes, and Glaudir Donato for always being my truest cheerleaders, as well as my nieces for inspiring me to work towards a better world for them.

I would like to acknowledge my peer mentor, Itamar Rocha, for giving me the best advice and bringing me key opportunities, and my friend and classmate, Humberto Navarro, for not leaving my side during all our courses and projects.

I would be remiss in not mentioning the Fundação Estudar for leveraging my career and my dreams while welcoming me into an inspiring community of Brazilians driven by impact.

Lastly, I would like to mention all my friends and labmates, without whom this journey would not have been nearly as rich and fun.

Resumo

Este trabalho introduz o problema do sequenciamento máximo de carros para apoiar as operações de montagem de uma multinacional automobilística. É proposta uma formulação de programação linear inteira (PLI) para sequenciar o maior número possível de carros sem violar restrições de espaçamento relacionadas a atributos especiais, como teto solar e rádio. Adicionalmente, limitantes superior e inferior, que podem ser calculados em tempos computacionais negligenciáveis, são apresentados, assim como algoritmos de buscas iterativa e binária para resolver o problema quando bons limites primais não estão prontamente disponíveis. Para obter soluções de boa qualidade rapidamente, desenvolveu-se um algoritmo de busca local iterada, cujas soluções são utilizadas para inicializar o resolvedor, aumentando a performance dos métodos exatos. Resultados computacionais demonstram *gaps* relativamente baixos para instâncias de *benchmark* em um tempo limite de 10 minutos. Além disso, foi conduzida uma análise do espaço de instâncias para identificar as características que dificultam a resolução do problema. Por fim, as demandas reais da empresa foram resolvidas em menos de um segundo, e simulações cronológicas, visando a maximizar o número de carros sequenciados por turno, foram conduzidas para quatro meses de dados históricos. Nesse caso, um novo modelo PLI foi empregado para sequenciar os carros não produzidos até o último turno do mês em uma linha dedicada, diminuindo o ritmo da linha de produção de forma a relaxar as restrições impostas pelos atributos especiais. Os resultados apontaram que a linha dedicada foi necessária em apenas um dos meses.

Palavras-chave: Escalonamento, Linha de produção, Sequenciamento de carros, Otimização combinatória.

Abstract

This work introduces the maximum length car sequencing problem to support the assembly operations of a multinational automotive company. We propose an integer linear programming (ILP) formulation to schedule the maximum number of cars without violating the so-called option constraints. In addition, we present valid combinatorial lower and upper bounds, which can be calculated in less than 0.01 seconds, as well as binary and iterative search algorithms to solve the problem when good primal bounds are not readily available. To quickly obtain high-quality solutions, we devise an effective iterated local search algorithm, and we use the heuristic solutions as warm start to further enhance the performance of the exact methods. Computational results demonstrate that relatively low gaps were achieved for benchmark instances within a time limit of ten minutes. We also conducted an instance space analysis to identify the features that make the problem more difficult to solve. Moreover, the instances reflecting the company's needs could be solved to optimality in less than a second. Finally, simulations with real-world demands, divided into shifts, were conducted over a period of four months. In this case, we use the proposed ILP model in all shifts except the last one of each month, for which we employ an alternative ILP model to sequence the unscheduled cars, adjusting the pace of the assembly line in an optimal fashion. The results pointed out that the latter was necessary in only one of the months.

Key-words: Scheduling, Assembly line, Car sequencing, Combinatorial optimization.

Contents

Glossary

ACO	:	Ant colony optimization
B&B	:	Branch-and-bound
BFS	:	Breadth-first search
BS	:	Beam search
CP	:	Constraint programming
CSP	:	Car sequencing problem
GA	:	Genetic algorithm
ILP	:	Integer linear programming
ILS	:	Iterated local search
ISA	:	Instance space analysis
LB	:	Lower bound
LNS	:	Large neighborhood search
MATILDA	:	Melbourne algorithm test instance library with data analytics
Max-CSP	:	Maximum length car sequencing problem
MIP	:	Mixed integer programming
SOS1	:	Special ordered set of type 1
UB	:	Upper bound
VNS	:	Variable neighborhood search

List of Figures

List of Tables

1 Introduction

1.1 Preliminaries

Car assembly operations have been widely studied in the literature, and relevant research focused on how product variety is necessary in the car industry even if it affects productivity (Fisher and Ittner, 1999). Therefore, workload allocations have to guarantee that operations of varying duration do not cause overtime on a paced assembly line. The motivation, welfare, and satisfaction of the employees are of paramount importance. Hence, workload assignments need to be perceived as doable and fair. An example of a scheduling solution aimed at improving employee satisfaction can be found in Bukchin and Masin [2004], and the need for balancing the work content on paced car assembly lines in order to avoid overtime has been discussed, for example, in Bhattacharyya et al. [1993].

This work is motivated by a collaboration with a large car assembly company in Brazil. Different models of cars are sequenced in the assembly line. No more than a maximum number of cars requiring a specific assembly operation should be scheduled in close proximity to one another so that the workers responsible for this assembly operation have enough time to complete the task without delaying the line. The need to quantify the complexity of operations in assembly lines has already been discussed in Zeltzer et al. [2017], and issues around ergonomics of diverse assembly operations were highlighted by McClellan et al. [2009].

The scheduling of car assembly operations has attracted interest since the definition of the *car sequencing problem* (CSP) by Parrello et al. [1986]. The issue of spacing out specific operations in the assembly sequence is a common constraint among papers focusing on this application. There seems to be consensus on modeling these constraints using the concept of *options*, which correspond to additional components to be assembled in a car, such as air-conditioning and sunroof. The stations responsible for assembling the options have limited workload capacity.

Figure 1 illustrates the issue with an example. Let the station that assembles the option sunroof have one worker and space for three cars at a time in the line. The cars move continuously at a fixed pace, and the time each worker takes to complete the task ranges from the moment the car gets into the station to the moment it leaves. Thus, if the worker from the sunroof station is already occupied with a car (car 3) when another one needing a sunroof (car 5) arrives, the latter will not be completed in time. For this case, an option constraint would establish that no more than one car requiring a sunroof should be scheduled in any subsequence of three consecutive cars.

The feasibility version of the CSP, as defined by Dincbas et al. [1988], requires

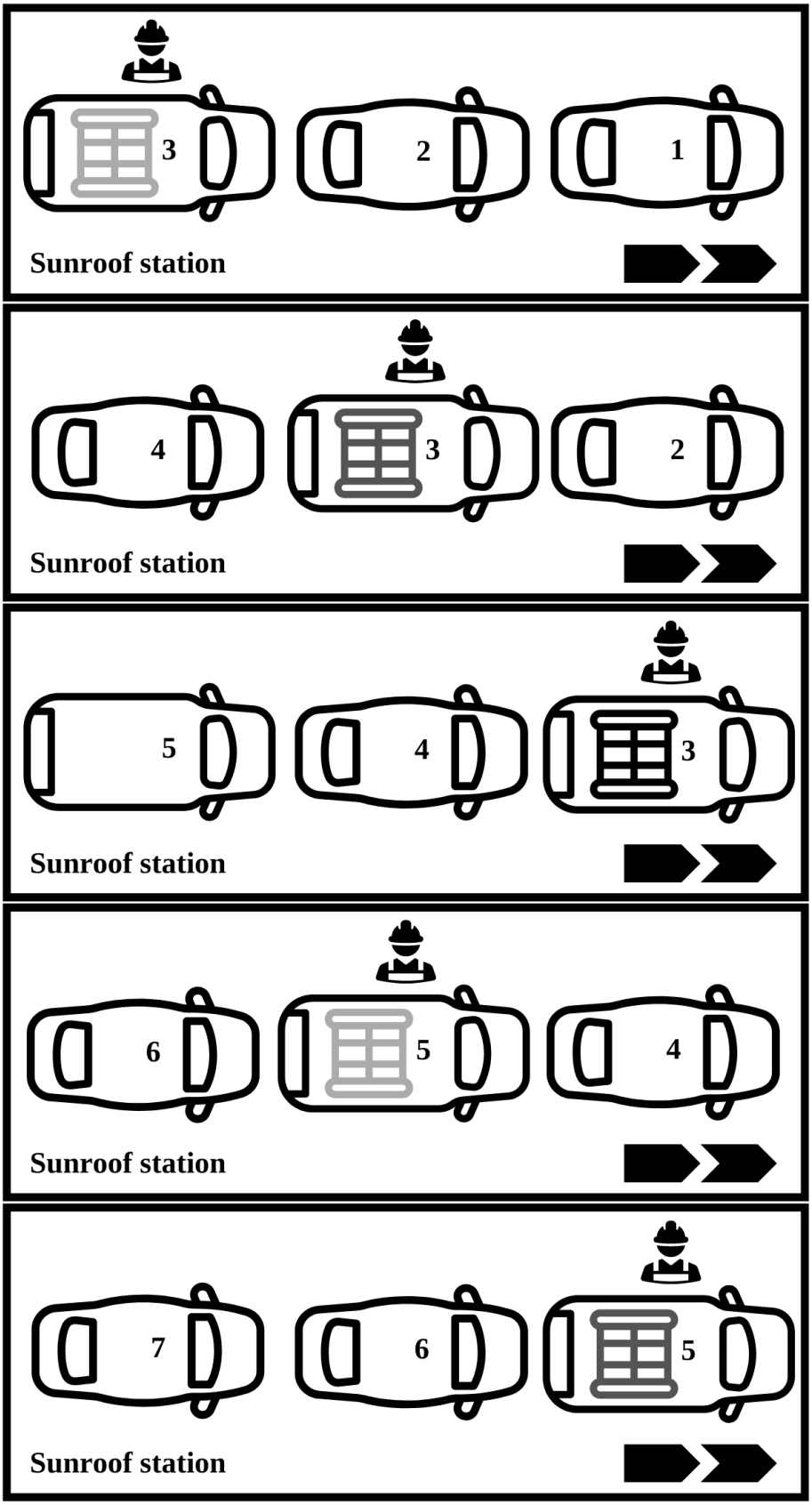

Figure 1: Example of assembly station.

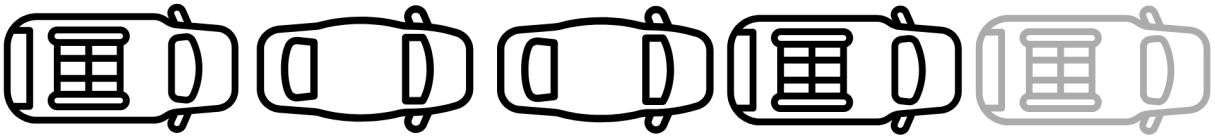

Figure 2: Example of a feasible sequence followed by a car that causes an option constraint violation.

identifying if an assembly sequence with all demanded cars exists without violating any option constraint. The optimization version of the CSP has been formalized using many different objective functions in the literature. However, the most common feature seems to be accepting the violation of the option constraints by allowing delays in the production line. The resulting optimization problem requires scheduling all cars while minimizing a weighted combination of some penalizations. This objective and its variants will be formalized in the following sections.

We propose a new optimization variant for the CSP, the *maximum length car sequencing problem* (Max-CSP). The maximum length refers to the number of cars scheduled contiguously in the assembly line, resulting in the longest sequence without option constraint violations. Figure 2 shows an example of a feasible sequence followed by a car that causes a violation. Even if the production demands an additional car with a sunroof, it cannot be scheduled without violating the option constraint that limits to one the maximum number of such cars every three consecutive positions. The motivations for developing this variant are detailed in the next section. This work aims at introducing and proposing solution methods for the Max-CSP.

1.2 Motivation

The partner company that motivated the definition of the Max-CSP manages its scheduling decisions by modeling only some of the "hard sequencing rules"(see Boysen et al., 2009). They consider a subset of the option constraints, and the options have all to be respected, as any violation leads to a stop of the whole assembly line. The assembly operations refer to the general assembly line (the last step before car completion), a single line runs in the factory, and the scheduling is solved daily. It is crucial to maintain the flow on the line smooth, fast, and without interruptions.

The management of the company explained that they would rather assemble unscheduled cars in the days after the originally planned one and add a dedicated shift at the end of the time horizon for those cars not yet assembled. Car demands have a dynamic nature, i.e., they arrive continuously, and must typically be assembled within a month. The cars not completed in this time frame are all scheduled during a dedicated shift, in which the pace of the line is slowed down. For that, we devise a different mixed *integer*

linear programming (ILP) formulation. This subjacent problem is named the Slow-CSP.

The company, prior to our collaboration, was allocating cars manually to the paced assembly line. This task is time-consuming, and often only solutions with option constraint violations could be found. The goal of our collaboration has been to develop an algorithm to solve this scheduling problem, identifying solutions without option violations while maximizing the number of cars assembled in the line and handling the unscheduled cars in the dedicated shift. The Max-CSP, as well as the Slow-CSP, are variants of the CSP (Parrello et al., 1986). However, we are not aware it has ever been studied in any of these forms.

The importance of this study is not only practical but also theoretical. The presented problem is complex, as it is to achieve good solutions. Thus, extensive research was conducted to generate integrated exact and heuristic methods that solve the Max-CSP effectively and efficiently.

1.3 Objectives

The objectives of this work are manifold and they are summarized as follows.

- Introduce a new variant of the CSP (Max-CSP), motivated by a real-life application, and describe how the problem can be positioned in the literature.
- Present valid combinatorial lower and upper bounds, which can be calculated in less than 0.01 seconds.
- Devise exact algorithms based on binary and iterative searches that yield good results when good primal bounds are not readily available.
- Implement an effective *iterated local search* (ILS) heuristic, with efficient intensification and diversification procedures, capable of finding high-quality solutions in a matter of seconds.
- Introduce a subjacent variant (Slow-CSP) to adjust the pace of the assembly line for the unscheduled cars in a dedicated shift.
- Conduct extensive computational experiments to evaluate the performance of all proposed exact and heuristic methods.
- Perform an instance space analysis to suggest which are the characteristics that make our problem challenging to solve.
- Simulate the scheduling decisions for a period of four months, using real-world instances.

1.4 Monograph outline

The remainder of this work is organized as follows. Section 2 provides a formal description of the problems and their relation with existing CSP variants, as well as a brief literature review. Section 3 models the Max-CSP, while Section 4 models the Slow-CSP. Section 5 describes the iterative and binary search algorithms for the Max-CSP, and Section 6 presents valid combinatorial bounds. Section 7 describes the proposed heuristic algorithm. Section 8 contains the results of the computational experiments. Finally, Section 9 summarizes our findings.

2 Context and related work

2.1 Description of the problems and literature positioning

In the following, we first address the feasibility version of the CSP and compare it with the feasibility version of the Max-CSP. We then present the most widely studied objective functions for the optimization variants of the CSP and narrow down our analysis to a problem variant that we can easily adapt to the Max-CSP. The optimization version of the Max-CSP is also described within the context of our partner company, alongside a brief discussion on the Slow-CSP, the other problem variant introduced in this work.

The feasibility version of the CSP, as introduced by Dincbas et al. [1988], consists in sequencing D cars, where D is also the number of positions available for sequencing them while satisfying the option constraints. An option $j \in C$ imposes that no more than p_j cars can be scheduled in any subsequence of length q_j. It is common practice to group cars in "classes"if they have the same set of options, as cars in the same class can be assigned interchangeably a position in the sequence without violating any option constraint. Classes are denoted $i \in P$, and there are d_i cars in each class. If an option $j \in C$ applies to a class of cars $i \in P$, then $c_{ji} = 1$, and 0 otherwise. For future reference, we summarize the notation in the following bullet points.

- C: the set of options, such as air-conditioning and sunroof;
- P: the set of car classes;
- D: the total number of cars and the corresponding number of available positions in the assembly line;
- (p_j, q_j): p_j defines the maximum number of cars allowed with option $j \in C$ in any subsequence of q_j consecutive positions;
- d_i: the demand for cars of class $i \in P$;
- c_{ji}: whether or not the cars of class $i \in P$ require the option $j \in C$.

The feasibility version of the CSP has been proven to be $\mathcal{NP}$-complete in the strong sense by Kis [2004], using a reduction from the Exact Cover by 3-sets. Note that Estellon and Gardi [2013] proved that the CSP is $\mathcal{NP}$-hard using a reduction from the *Hamiltonian path problem* with even more restrictive options.

In the Max-CSP, we study the same feasibility problem as for the CSP, considering all options (as for the CSP), including the subset of options currently implemented by the company, namely:

- $j \in C \mid (p_j = 1, q_j)$ states that at most one car with option j can be scheduled in any subsequence of length q_j;
- $j \in C \mid (p_j = q_j - 1, q_j)$ states that at most $q_j - 1$ cars with option j can be scheduled in any subsequence of length q_j.

The feasibility version of the Max-CSP limited to these options is a special case of the CSP. However, the complexity proofs by Kis [2004], Estellon and Gardi [2013] are valid for this variant as well, as the options used in the transformation match those discussed above. Hence, the feasibility version of the Max-CSP is $\mathcal{NP}$-complete in the strong sense both in the situation in which all options are imposed, as well as in the subset described above.

The optimization version of the CSP has attracted a lot of interest from the research community, and many variants have been studied in the past. We summarize the most widely studied optimization problems in this area of research, focusing on the CSP objective functions as this is the main difference between the Max-CSP and the CSP.

In their literature review, Boysen et al. [2009] classify the CSP according to different objective functions, which aim at minimizing:

- the number of subsequences (windows) with at least one violation of the option constraints, using a "sliding windows"approach (e.g. Gottlieb et al., 2003, Gravel et al., 2005, Golle et al., 2015, Yilmazlar and Kurz, 2023);
- the number of subsequences starting with a car under a violated option constraint (e.g. Boysen and Fliedner, 2007, Fliedner and Boysen, 2008);
- the number of exceeding cars with a violated option constraint, for every window (e.g. Bolat and Yano, 1992; Hottenrott et al., 2021);
- the value of a penalty function based on individual weights, given as input data, for each additional option violation (e.g. Parrello et al., 1986, Smith, 1997);
- the number of empty slots necessary to generate a feasible solution (e.g. Hindi and Ploszajski, 1994, Perron and Shaw, 2004).

Other relevant objective functions are the following.

- A lexicographic multi-objective function that considers the assembly shop (minimizing the violation of priority and non-priority ratio constraints) and the paint shop (minimizing the color changes in the line). This is the objective function presented by Renault for the ROADEF'2005 Challenge [Solnon et al., 2008]. Wu et al. [2021]

and Zhang and Ding [2023] include a paint body storage between the paint shop and the assembly shop, thus utilizing a multi-stage approach.

- The minimization of the exceeding cars with a violated option constraint, for every window and including weights (denoted "upper over-assignment"), and the weighted count of the times an option constraint is loosely satisfied (the count of how many cars are scheduled less than the maximum requirement set by each option constraint and for each window, denoted "upper under-assignment"). This objective function was first proposed by Bautista et al. [2008] and further explored in later works, such as Thiruvady et al. [2011, 2014, 2020].

- The minimization of unscheduled options in the assembly line [Souza et al., 2023], so as to minimize overload in a subsequent day. This recent definition is the closest to ours found in the literature. It can lead to a shorter sequence than maximizing the scheduled cars in a given shift, which is the main objective of our partner company, but future work may explore its performance in this industrial setting.

As for the Max-CSP, the optimization version maximizes the length of the assembly line operation by sequencing the maximum number of cars without violating any option constraint. In practice, if any car does not fit into the sequence (as it would violate one or more than one option constraints), it is re-scheduled on the subsequent shift.

To better illustrate the sequencing process in a given shift, consider the following instance. Let $P = \{1, 2, 3, 4\}$ be the set of classes, and the corresponding demands for each class be $d_1 = 1$, $d_2 = 2$, $d_3 = 3$, and $d_4 = 1$. The options are $(p_1 = 1, q_1 = 3)$ with $c_{12} = c_{14} = 1$, and $(p_2 = 2, q_2 = 3)$ with $c_{23} = c_{24} = 1$. Figure 3 shows a valid sequence for this instance in solid lines. Since a car of class 4 was scheduled right after a car of class 3, adding another car of class 3 at the end of the sequence would violate the constraint associated with option 2 (i.e., no more than 2 cars with option 2 can be scheduled in a window of 3 positions). Even if it was possible to schedule such a car in that position, appending a car of class 2 to the new solution would violate the constraint associated with option 1 (i.e., at most 1 car with option 1 in a window of 3 positions). In that case, the remaining cars of classes 2 and 3 would be left outside the sequence.

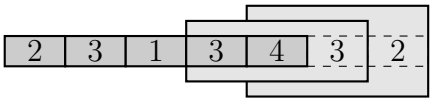

Figure 3: Example of a sequencing process.

At the end of the production time horizon (typically a month), the company runs one additional shift at a slower pace of the line, so as to schedule the cars that did not fit into the last shift. For the latter, we define the Slow-CSP, which finds the minimum

pace delay that loosens the option constraints just as much as it is necessary to fit all cars. This problem is solved for a single shift within the time horizon determined by the company, so it is not the main focus of this work. Nevertheless, it is modeled in Section 4.

More variants associated with constraints on the number of options allowed, constraints on production cycles and associated options assignments, and constraints on the number of cycles options are valid for can be reviewed in Boysen et al. [2009], but they do not relate to the Max-CSP. Similarly, we refer the reader to Yavuz and Ergin [2018] for a reference on papers integrating car sequencing and level scheduling problems.

One of the most popular variants of the car scheduling problem is the one studied by Bautista et al. [2008], which is still attracting wide attention from the research community in recent years and is formalized in the following section. Benchmark instances and recent metaheuristics are publicly available. The mathematical formulation for this variant allows for adaptations to the Max-CSP, which will be discussed later.

2.2 The CSP by Bautista et al. [2008]

In Bautista et al. [2008], the definition of option constraints is the same as the previous literature, however, a penalization is accounted for in the objective function if the constraints are violated or if they are loosely satisfied. Additional notation is required to formally introduce the objective function and the mathematical model proposed by the authors, namely:

- the upper over-assignment is $y_{jt} = \max\{0, \sum_{k=l_j(t)}^{t} c_{j,u(k)} - \min\{t, p_j\}\}, \forall j \in C, t = 1, \ldots, D$ and $l_j(t) = \max\{1, t+1-q_j\}$. This term represents the positive difference between the number of times an option appears in the subsequence $[l_j(t), t]$ and the maximum number it should appear, p_j (i.e., the number of violations of option j in the subsequence ending in t). This value is bound to be in the interval $[0, q_j - p_j)$.
- α_{jt} the weight assigned to each unit of violation $y_{jt}, \forall j \in C, t = 1, \ldots, D$.
- the upper under-assignment is $z_{jt} = \max\{0, \min\{t, p_j\} - \sum_{k=l_j(t)}^{t} c_{j,u(k)}\}, \forall j \in C, t = 1, \ldots, D$. This term represents the positive difference between the number of times p_j an option could appear and the number of times the option appears in the subsequence $[l_j(t), t]$ (i.e., the number of times an option j in the subsequence ending in t is loosely satisfied). This value is bound to be in the interval $[0, p_j)$.
- β_{jt} the weight assigned to each unit of $z_{jt}, \forall j \in C, t = 1, \ldots, D$.
- $x_{it} = 1$ if a car of class $i \in P$ is scheduled in position $t = 1, \ldots, D$, 0 otherwise.

The objective function and the mathematical model proposed by Bautista et al. [2008] can now be stated as

$$\min \quad \sum_{t=1}^{D} \sum_{j \in C} (\alpha_{jt} y_{jt} + \beta_{jt} z_{jt}) \tag{1}$$

$$\text{subject to} \quad \sum_{i \in P} x_{it} = 1 \qquad t = 1, \ldots, D \tag{2}$$

$$\sum_{t=1}^{D} x_{it} = d_i \qquad i \in P \tag{3}$$

$$z_{jt} - y_{jt} + \sum_{i \in P} \sum_{k=l_j(t)}^{t} c_{ji} x_{ik} = \min\{t, p_j\} \qquad j \in C, t = 1, \ldots, D \tag{4}$$

$$y_{jt}, z_{jt} \geq 0 \qquad j \in C, t = 1, \ldots, D \tag{5}$$

$$x_{it} \in \{0, 1\} \qquad i \in P, t = 1, \ldots, D. \tag{6}$$

The objective function (1) minimizes the weighted combination of upper over-assignment and upper under-assignments. Constraints (2) guarantee that every position has exactly one car scheduled. Constraints (3) ensure the class demands are met. Option constraints are imposed in Constraints (4), and the domain of the variables in Constraints (5) and (6).

2.3 Literature review

We provide a brief literature review including exact and heuristic approaches previously studied for solving some of the most relevant versions of the CSP.

For the feasibility version, several methods are explored, such as a branching scheme along with bounding algorithms [Drexl et al., 2006], *pure boolean satisfiability* [Artigues et al., 2014], and *constraint programming* (CP) [Artigues et al., 2014, Siala et al., 2015].

For the optimization version focused on minimizing violations in the assembly shop, Warwick and Tsang [1995] devise a *genetic algorithm* (GA), Gravel et al. [2005] utilize an ILP model and *ant colony optimization* (ACO), and Fliedner and Boysen [2008] describe a *branch-and-bound* (B&B) algorithm.

Solnon et al. [2008] review the scientific production related to the ROADEF'2005 challenge. The winning entry, Estellon et al. [2008], integrated a very *large-scale neighborhood search* (LNS) with an ILP formulation and fast explorations of small neighborhoods. Ribeiro et al. [2008] were ranked second, employing an ILS and a *variable neighborhood search* (VNS). The local search strategies developed by the first and second places

contain swap and insertion movements, which are similar to the ones presented in Section 7.

Including a buffer between the paint shop and the assembly shop, in Guerre-Chaley et al. [1995], cars have a due date to be assembled, and the demand is processed car by car. Wu et al. [2021] propose two metaheuristics based on *tabu search* and VNS, the latter embedded in a nested partition framework. The work by Zhang and Ding [2023] has a dynamic nature and implements a GA and a greedy algorithm.

Moya et al. [2019] introduce a robust solution for uncertain partial demand of special vehicles in different production plans using a *greedy randomized adaptive search procedure*, ACO, and VNS. Hottenrott et al. [2021] combine a B&B algorithm with an adaptive LNS heuristic to tackle robustness against short-term sequence alterations.

Lastly, the works that refer to the formulation by Bautista et al. [2008] are discussed in the following. Bautista et al. [2008] solve instances with up to 400 cars using a *beam search* (BS) algorithm. Thiruvady et al. [2011] solve instances with up to 600 cars hybridizing BS, ACO, and CP. Thiruvady et al. [2014] conclude that a *hybrid Lagrangian relaxation* ACO algorithm is the best algorithm among those tested for instances with up to 300 cars. Moreover, Thiruvady et al. [2020] employ a VNS in which an appropriately long subsequence of cars is re-optimized using an ILP model (and variable fixing for the rest of the sequence). Instances with up to 500 cars are considered. This method compares favorably to the literature both in terms of solution quality and CPU times. Finally, Sun et al. [2023] identify critical features that make instances harder to solve. Their principal component analysis investigation shows that high utilization of options and large average number of options per car class are the critical features. Furthermore, the authors use machine learning to choose the best method for an instance among the algorithm by Thiruvady et al. [2014], an improved version of the VNS by Thiruvady et al. [2020], and the ILP by Bautista et al. [2008] in which Constraints (4) are added dynamically via a "lazy"separation procedure. The improved ILP performs best for the hardest instances, and the ILP with lazy constraints is the best-performing method for medium-difficulty instances.

3 Modeling the Max-CSP

In what follows, we present two possible ways of mathematically modeling the Max-CSP.

3.1 Using the mathematical formulation by Bautista et al. [2008] to solve the Max-CSP

The Max-CSP aims at maximizing the length of the sequence without any options violation. Our first attempt at solving the Max-CSP exactly is that of adapting a model from the literature. In particular, the Bautista et al. [2008] model can be converted to solve the Max-CSP as follows.

Let π^* be an optimal sequence for the Max-CSP. If $|\pi^*| = D$, the model of Bautista et al. [2008] returns the optimal Max-CSP solution. However, if $|\pi^*| < D$, given the corresponding CSP solution, let t^* be the position at which the first option violation occurs. In other words, the number of option constraint violations in the interval $[l_j(t), t]$ is $y_{jt} = 0$ for all $j \in C$ and all $t < t^*$, and $y_{jt^*} > 0$ for some $j \in C$. The partial CSP solution from position $t = 1$ to position $t^* - 1$ is a feasible solution for the Max-CSP.

Proposition 1. *If we set*

$$\alpha_{jt} = \begin{cases} 1 & \text{if } t = D \\ 1 + \sum\limits_{m \in C} \sum\limits_{k=t+1}^{D} \alpha_{mk}(q_m - p_m) & \text{if } 1 \leq t < D \end{cases}$$

and $\beta_{jt} = 0, \forall j \in C, t = 1, \ldots, D$*, an optimal solution of Bautista et al. [2008] can be converted into an optimal solution for the Max-CSP, in which* $t^* = |\pi^*| + 1$*.*

Proof. A solution of the Max-CSP is a valid CSP solution when all unscheduled cars are appended at the end of the violation-free sequence in any given order. If a CSP solution existed where $t^* \geq |\pi^*| + 1$, this solution could be converted into a sequence with no violations and whose length is greater than $|\pi^*|$, contradicting the initial assumption that π^* is optimal for the Max-CSP. Hence, the first violation in an optimal sequence for the CSP must occur at a position $t^* \leq |\pi^*| + 1$.

To show that t^* cannot be strictly less than $|\pi^*| + 1$, note that, for any CSP instance, the maximum value that the variable y_{jt} may achieve for all $j \in C$, $t = 1, \ldots, D$,

is bounded from above by $q_j - p_j$. As such, in any solution, the inequalities

$$\alpha_{jt} > \sum_{m \in C} \sum_{k=t+1}^{D} \alpha_{mk}(q_m - p_m) > \sum_{m \in C} \sum_{k=t+1}^{D} \alpha_{mk} y_{mk}$$

hold for all $j \in C$, $t = 1, \ldots, D$. Hence, if $t^* < |\pi^*| + 1$, the objective value of the CSP solution is worse than the one provided by the Max-CSP solution π^*, and thus it is not optimal, contradicting again our hypothesis.

Since t^* is neither strictly less than $|\pi^*| + 1$ nor strictly greater than $|\pi^*| + 1$, it must be exactly equal to $|\pi^*| + 1$.

Then, for all $j \in C$, we define α_{jt} as

$$\alpha_{jt} = \begin{cases} 1 & \text{if } t = D \\ 1 + \sum_{m \in C} \sum_{k=t+1}^{D} \alpha_{mk}(q_m - p_m) & \text{if } 1 \leq t < D \end{cases}$$

where the maximum possible value of α_{jt} is $\mathcal{O}([\sum_{m \in C}(q_m - p_m)]^D)$, and $\beta_{jt} = 0, \forall j \in C, t = 1, \ldots, D$.

This choice of coefficients makes any violation in an earlier position of the sequence more costly than any violation in all the subsequent positions of the sequence ($\alpha_{jt} > \sum_{m \in C} \sum_{k=t+1}^{D} \alpha_{mk}, \forall t = 1, \ldots, D$). Therefore, the objective function will avoid violations in early positions, if feasible. The resulting Max-CSP solution can be obtained by retaining all cars sequenced from position 1 until the first violation encountered in the CSP solution, if any.

□

The exponential growth of the penalization coefficients limits the implementation of the model by Bautista et al. [2008]. In the instances provided by the company (with six options), the penalization coefficients could be encoded on a double variable in the *mixed integer programming* (MIP) solver (CPLEX) for instances with up to 250 cars. Therefore, we propose a new formulation that does not rely on penalization coefficients, as described in the following.

3.2 Proposed formulation

Let $x_{it} = 1$ if a car of class $i \in P$ is scheduled in position $t = 1, \ldots, D$, 0 otherwise. The proposed formulation (F1) for the Max-CSP reads as follows:

$$\text{(F1)} \quad \max \quad \sum_{i \in P} \sum_{t=1}^{D} x_{it} \tag{7}$$

$$\text{subject to} \quad \sum_{i \in P} x_{it} \leq 1 \qquad t = 1, \ldots, D \tag{8}$$

$$\sum_{i \in P} x_{it+1} \leq \sum_{i \in P} x_{it} \qquad t = 1, \ldots, D-1 \tag{9}$$

$$\sum_{t=1}^{D} x_{it} \leq d_i \qquad i \in P \tag{10}$$

$$\sum_{i \in P} \sum_{k=l_j(t)}^{t} c_{ji} x_{ik} \leq p_j \qquad j \in C, t = 1, \ldots, D \tag{11}$$

$$(6).$$

The objective function aims to maximize the number of cars scheduled. Constraints (8) and (9) ensure that every position is associated with at most one car and that the empty positions are at the end of the sequence, respectively. Constraints (10) guarantee that the demand for each class is not surpassed. Lastly, option constraints are imposed in Constraints (11).

The formulation F1 can be solved with a commercial solver. However, its performance is highly dependent on the availability of good upper and lower bounds. Preliminary tests also showed that the feasibility version proved much easier to solve. Hence, we tested alternative exact approaches in the spirit of Silva et al. [2023], which implemented an iterative strategy and a binary search via feasibility checking to enhance performance in the context of a parallel machine scheduling problem. For the Max-CSP, we developed a branching scheme on the first option violation and iterative methods based on fixing the number of cars scheduled in the solution. In the following, we describe the iterative methods (an incremental search, a decremental search, and a binary search) and the branching strategies (considering alternative prioritization).

4 Modeling the Slow-CSP

At the end of a production cycle (typically a month), some cars might not have been scheduled. The managers schedule these cars on a dedicated shift, demanding a feasible sequence with all cars. Therefore, the managers reduce the assembly line speed in order to give enough time for the assembly operations to be completed without violating any option constraint. We introduce another variable, $s \in \mathbb{R}^*$, which sets the relative pace decrease of the assembly line (e.g., $s = 100\% = 1$ if the assembly line must be at half of the original pace to allow all the operations to be done in time, as double the operations can be performed in each station). The Slow-CSP formulation reads as follows:

$$\begin{aligned}
\min \quad & s && && (12)\\
\text{subject to} \quad & \sum_{i \in P} x_{it} = 1 && t = 1, \ldots, D && (13)\\
& \sum_{t=1}^{D} x_{it} = d_i && i \in P && (14)\\
& \sum_{i \in P} \sum_{k=l_j(t)}^{t} c_{ji} x_{ik} \leq p_j(s+1) && j \in C, t = 1, \ldots, D && (15)\\
& s \in \mathbb{R}^* && && (16)\\
& (6). &&
\end{aligned}$$

The objective function minimizes the pace decrease of the assembly line. Constraints (13) guarantee that every position has exactly one car scheduled. Constraints (14) ensure the class demands are met. Constraints (15) relax the option constraints proportionally to the pace decrease. By solving this model, we obtain both the optimal pace and the associated sequence for the slower shift.

5 Exact strategies based on iterative search methods for the Max-CSP

We develop exact algorithms based on iteratively solving the feasibility CSP problem. This problem can be formulated as

$$\begin{aligned} \text{(I)} \quad \max \quad & 0 && && (17)\\ \text{subject to} \quad & \sum_{i \in P} x_{it} = 1 && t = 1, \ldots, D^* && (18)\\ & (6), (10), (11). \end{aligned}$$

The optimal solution for the Max-CSP can be obtained by testing all values of $D^* = \{1, \ldots, D\}$ and picking the maximum one leading to a feasible solution. The value of D^* can be bounded from above and below, based on the availability of upper and lower bounds to speed up the search. Furthermore, we can use alternative exploration strategies, namely:

- I_I an iterative incremental approach. As can be observed in Algorithm 1, given a feasible solution π', its corresponding primal bound lb, and a dual bound ub, the algorithm iteratively checks if it is feasible to schedule one more car (Lines 3–4). If so, the best solution is updated (Lines 5–6). In case it is not feasible, $lb = ub$ or the time limit is reached, the main loop is interrupted, and the procedure returns the best solution (Line 7).

- I_D an iterative decremental approach. Algorithm 2 presents the steps for the decremental search, which receives a primal solution π', its associated bound lb, and a dual bound ub. Starting from ub, at each iteration, the feasibility must be proven false so the decrement can be performed (Line 5). If the sequence is feasible, it is then optimal, and the solution is returned (Lines 3–4). If no feasible solution is found within the time limit, the primal solution passed as input is returned (Line 6).

- I_B a binary search. Algorithm 3 shows the pseudocode of the binary search. The algorithm receives as input a feasible solution π' along with its associated primal bound lb, as well as a valid dual bound ub. At each iteration, the binary search takes the middle element of lb and ub, and assigns the resulting value to a variable m (Line 2). Next, the procedure checks whether it is possible to build a feasible sequence with exactly m cars (Line 3). If so, that becomes the new best feasible solution, and the lb is updated to m (Lines 4–6). Otherwise, ub is set to $m - 1$

(Line 8). The loop continues until the two bounds are the same or the time limit is reached, and the best solution found is returned (Line 9).

Algorithm 1: IncrementalSearch(π', lb, ub)

```
1 feasible ← true
2 while feasible = true and lb ≤ ub and TimeLimitNotReached do
3 |   lb ← lb + 1
4 |   [π, feasible] ← CheckFeasibility(lb)
5 |   if feasible = true then
6 |   |   π' ← π
7 return π'
```

Algorithm 2: DecrementalSearch(π', lb, ub)

```
1 while feasible = false and lb < ub and TimeLimitNotReached do
2 |   [π, feasible] ← CheckFeasibility(ub)
3 |   if feasible = true then
4 |   |   return π
5 |   ub ← ub − 1
6 return π'
```

Algorithm 3: BinarySearch(π', lb, ub)

```
1 while lb < ub and TimeLimitNotReached do
2 |   m ← ⌈(lb + ub)/2⌉        ▷ middle element of lb and ub, rounded up
3 |   [π, feasible] ← CheckFeasibility(m)
4 |   if feasible = true then
5 |   |   π' ← π
6 |   |   lb ← m
7 |   else
8 |   |   ub ← m − 1
9 return π'
```

These strategies require the implementation of *ad hoc* algorithms. It is possible to use CPLEX embedded functions to explore similar strategies if we introduce a new variable z_t in formulation F1. Variable z_t is equal to 1 if t is the first unoccupied position in the sequence. Thus, $z_t = 1$ if $t = D + 1$, $z_0 = 0$ otherwise. The resulting mathematical model is:

$$\text{(F2) max} \quad \sum_{t=2}^{D+1} (t-1) z_t \tag{19}$$

$$\text{subject to} \quad \sum_{i \in P} x_{it} + \sum_{k=1}^{t} z_k = 1 \qquad t = 1, \ldots, D \tag{20}$$

$$\sum_{t=1}^{D+1} z_t = 1 \tag{21}$$

$$z_t \in \{0, 1\} \qquad t = 1, \ldots, D+1 \tag{22}$$

$$(6), (10), (11).$$

F2 is an alternative valid formulation for the Max-CSP, and we can take advantage of the special ordered set of type 1 (SOS1) branching strategies embedded in CPLEX, given Constraints (21).

The variables in SOS1 have to be assigned unique weights in CPLEX, corresponding to the branching priority of each variable. Therefore, we tested F2 without SOS1 while still enforcing a higher branching priority to the z_t variables, with SOS1 assigning incremental weights to the z_t variables (F2_I), assigning decremental weights to the z_t variables (F2_D), and mimicking with weights the priorities imposed by a binary search (F2_B). Algorithm 4 presents the steps for setting these weights through a *breadth-first search* (BFS).

The weights are assigned in descending order to those positions, as they are associated with a branching priority in the corresponding variables z_t. The algorithm receives the lower and upper bounds (lb' and ub') of the interval where the binary search would be performed. The variables of type SOS1 within the same set in CPLEX require their weights to be unique. Therefore, one starts assigning the smallest weights to the positions outside the bounds of the binary search (Lines 3–5). Next, to work as a flag for the positions unvisited by the BFS, the weights inside the bounds are assigned value 0 (Lines 6–7). Finally, the last loop of the algorithm performs the BFS (Lines 10–19), which iteratively stores the lower and upper bounds of the search intervals in a queue. Every time a pair of bounds is popped from the queue (Line 11), their middle element is calculated (Line 12). If the corresponding position has not been visited yet, its weight receives the next largest value (Line 15), and the search interval is divided into two, which are pushed into the queue (Lines 17–19). Lastly, the list of weights γ is returned (Line 20).

Algorithm 4: SetBinaryWeights(lb', ub')

```
g ← 0
γ ← list of size D + 1
for t = 1, ..., lb' − 1, ub' + 1, D + 1 do
    γ_t ← g
    g ← g + 1
for t = lb', ..., ub' do
    γ_t ← 0
g ← D
queue ← enqueue(lb', ub')
while queue ≠ ∅ do
    (lb, ub) ← dequeue(queue)
    mid ← ⌈(lb + ub)/2⌉
    if γ_mid > 0 then
        continue
    γ_mid ← g
    g ← g − 1
    if lb < ub then
        queue ← enqueue(lb, mid − 1)
        queue ← enqueue(mid, ub)
return γ
```

6 Valid bounds for the Max-CSP

In order to obtain the bounds necessary for the iterative methods, we describe a combinatorial upper bound (UB) and a combinatorial lower bound (LB) for the Max-CSP.

6.1 A valid UB

For all $j \in C$, let V_j be the set of cars with option j that are not scheduled in the optimal solution. We cannot compute *a priori* the values of $V_j, j \in C$. However, we can estimate an LB $Y \leq \left|\bigcup_{j \in C} V_j\right|$. By definition, $\sum_{i \in P} d_i - Y$ is a valid UB.

We define the surplus of an option j as the number of cars with j that do not fit in any feasible schedule due to the associated option constraint. This is similar to the option utilization defined by Perron and Shaw [2004], but, instead of computing the length of the sequence necessary to schedule a given number of cars with option j, we calculate the number of cars with option j not fitting in a sequence of fixed length.

Given option $j \in C$, its surplus S_j is given by

$$S_j = \max\left\{0, \sum_{i \in P} c_{ji} d_i - \left(p_j \left\lfloor \frac{D}{q_j} \right\rfloor + \min\{p_j, D \bmod q_j\}\right)\right\}. \tag{23}$$

We provide a way to estimate Y by using the set of variables

$$Z_j = \begin{cases} S_j & \text{if } j = 1, \\ \max\{0, S_j - \sum_{m=1}^{j-1} \min\{Z_m, \sum_{i \in P} c_{ji} c_{mi} d_i\}\} & \text{otherwise.} \end{cases}$$

Proposition 2.

$$Y = \sum_{j \in C} Z_j \leq \left| \bigcup_{j \in C} V_j \right|.$$

Proof. We use induction on the number l of options in C.

For the base case, let $C = \{1\}$. The value given by the formula is $S_1 \leq |V_1|$.

For the induction hypothesis, we assume $W_1, \ldots, W_l, l = |C|$, are pairwise disjoint subsets such that for all $j \in C$, $W_j \subseteq V_j$, $Z_j = |W_j|$. If it holds when adding an extra option to C, the sum of the variables Z_j will be

$$\sum_{j=1}^{l+1} Z_j = \left| \bigcup_{j=1}^{l+1} W_j \right| \leq \left| \bigcup_{j=1}^{l+1} V_j \right|.$$

Z_{l+1} can be either equal to or greater than zero. In the former case, it follows that $W_{l+1} = \emptyset \subseteq V_{l+1}$. Conversely, if $Z_{l+1} > 0$ and considering that $S_{l+1} \leq |V_{l+1}|$, then

$$Z_{l+1} = S_{l+1} - \sum_{j=1}^{l} \min\{Z_j, \sum_{i \in P} c_{l+1,i} c_{ji} d_i\}$$

$$\leq |V_{l+1}| - \sum_{j=1}^{l} \min\{Z_j, \sum_{i \in P} c_{l+1,i} c_{ji} d_i\}. \tag{24}$$

Now, consider the set

$$\Omega = V_{l+1} \setminus \left(\bigcup_{j=1}^{l} W_j \right),$$

whose cardinality may be computed as

$$|\Omega| = |V_{l+1}| - \sum_{j=1}^{l} |V_{l+1} \cap W_j|. \tag{25}$$

Comparing (24) and (25), since $Z_j = |W_j| \geq |V_{l+1} \cap W_j|$ and $\sum_{j=1}^{l} \sum_{i \in P} c_{l+1,i} c_{ji} d_i \geq |V_{l+1} \cap W_k|$ for all $k = 1, \ldots, l$, it follows that $Z_{l+1} \leq |\Omega|$. Thus, there must be at least one set $W_{l+1} \subseteq \Omega$ such that for all $j = 1, \ldots, l$, $W_{l+1} \cap W_j = \emptyset$, $|W_{l+1}| = Z_{l+1}$. □

6.2 A valid LB

New cars may be easily added to a given feasible sequence as long as: (i) they are given enough space between them when placed into the sequence so as to not violate their option constraints; and (ii) their options do not conflict with those in the sequence. In view of this, we propose a procedure to compute a valid LB iteratively. Let

$$G_j = \frac{q_j - p_j}{p_j}.$$

We define

$$H_i = \max_{j \in C} \{c_{ji} G_j\}$$

as a score associated with the most restrictive option of class $i \in P$. Furthermore, let Q_i be the set of cars of class i and

$$T(h) = \bigcup_{i \in P | H_i = h} Q_i$$

be the set of cars whose score is h. For the sake of simplicity, we use H with both classes and T sets interchangeably (i.e., $v \in Q_i \in T \implies H_i = H_T$). We also denote $\Pi_T = T_0, \ldots, T_{|C|}$ as a permutation of sets T such that $H_{T_o} < H_{T_{o+1}}$, for all $o = 0, \ldots, |C| - 1$ (i.e., Π_T is sorted in ascending order of scores). In addition, for each $j \in C$, we have a set $U(j)$ consisting of cars $v \in Q_i, \forall i \in P$, where $c_{ji} = 1$, that is, $U(j)$ contains the cars that have option j. Similarly to Π_T, $\Pi_U = U_1, \ldots, U_{|C|}$ is a permutation of the sets U such that $H_{T_o} = G_{U_o}$. Lastly, the spacing of the option linked to G_{U_o} can be represented as p_{U_o} and q_{U_o}.

We compute an LB recursively by using the set of variables L_o, which are calculated as shown in Equation (26). Furthermore, in order to keep track of how many cars in set U_r, $r = 1, \ldots, |C|$, were added to the solution up to iteration o, we define the set of variables A_{ro}, which are computed as in Equation (28).

$$L_o = \begin{cases} |T_o|, & \text{if } o = 0, \\ L_{o-1} + A_{oo}, & \text{if } o > 0. \end{cases} \tag{26}$$

$$\theta(U, L) = p_U \left(\left\lfloor \frac{L}{q_U - p_U} \right\rfloor + 1 \right) \tag{27}$$

$$A_{ro} = \begin{cases} \max\{\min\{\theta(U_o, L_{o-1}), |T_o|\} - \sum_{u=1}^{o-1} \min\{|T_o \cap U_u|, A_{u,o-1}\}, 0\} & \text{if } r = o, \\ A_{r,o-1} + \min\{|T_o \cap U_r|, A_{oo}\} & \text{if } 1 \leq r < o. \end{cases} \tag{28}$$

Proposition 3. *$L_{|C|}$ is a valid LB.*

Proof. The proof is based on the notion that, for each o, we can construct a new solution π_o by inserting cars from T_o into it in a specific way. For that, we use induction in o. For the base case $o = 0$, note that a sequence with all cars in T_0 may be easily constructed by sequencing them in any order since they have no associated option constraints. Therefore, $L_0 = |T_0|$ is a valid LB.

For the inductive step, suppose π_{o-1} is a sequence with L_{o-1} cars from $\bigcup_{r=0}^{o-1} T_r$, with $o > 0$. Additionally, assume that for each $r = 0, \ldots, o-1$, $A_{r,o-1}$ is an UB on the number of cars in U_r which are in π_{o-1}. If $T_o \cap \bigcup_{r=0}^{o-1} T_r = \emptyset$, the number of cars in T_o which may be added to π_{o-1} by correctly spacing them according to their most restrictive option is

$$\min\{\theta(U_o, L_{o-1}), |T_o|\}.$$

This can be seen in Figure 4, which illustrates how to interpose cars in π_{o-1} with the ones in T_o depending on their option constraints.

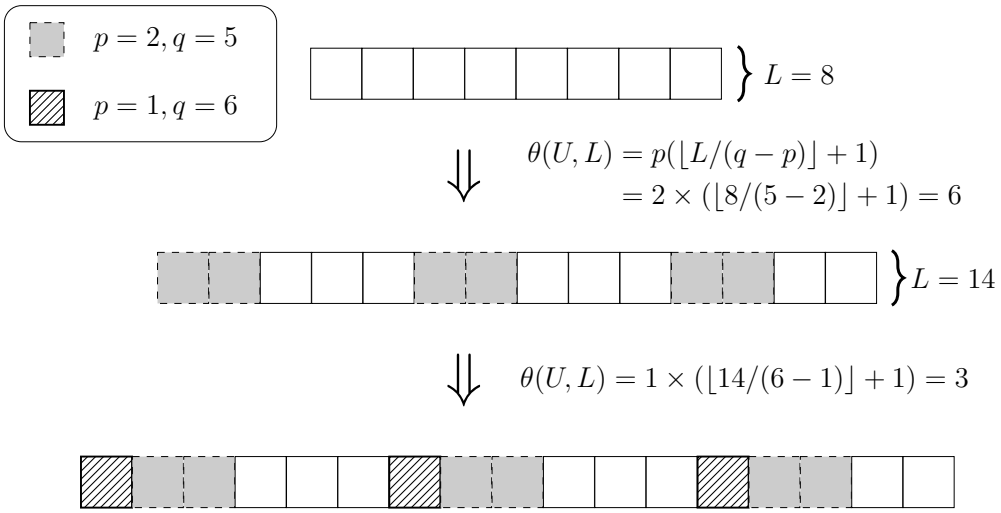

Figure 4: Car interposition based on the option constraints.

Now, suppose there exists at least one set U_r such that $T_o \cap U_r \neq \emptyset$, $r < o$. This means some cars in π_{o-1} may share the option from U_r with cars in T_o. Since at most

$A_{r,o-1}$ cars in U_r are in π_{o-1}, we may safely assume that we can subtract

$$\min\{|T_o \cap U_r|, A_{r,o-1}\}$$

from the total amount of new cars added to avoid possible conflicts. Thus, adding

$$A_{oo} = \max\{\min\{\theta(U_o, L_{o-1}), |T_o|\} - \sum_{u=1}^{o-1} \min\{|T_o \cap U_u|, A_{u,o-1}\}, 0\}$$

cars in T_o to π_{o-1} will always lead to feasible solution.

Finally, suppose π_o is a new solution, constructed by adding A_{oo} cars from T_o to π_{o-1}. Clearly, for all $r = 0, \ldots, o-1$, if X is the amount of cars in U_r which were added to π_{o-1} to construct π_o, then, $X \leq |T_o \cap U_r|$ and $X \leq A_{oo}$. Hence,

$$A_{ro} = A_{r,o-1} + \min\{|T_o \cap U_r|, A_{oo}\}$$

is a valid UB to the number of cars in U_r which were added to π_o. □

A primal solution associated with the LB can be obtained by initializing π with T_0, then inserting $L_o - L_{o-1}$ cars from T_o in π for each $o = 1, \ldots, |C|$, as shown in Algorithm 5. The insertion of each car needs to be performed in specific positions to guarantee the solution's feasibility. First, one makes sure there are cars to be inserted during iteration o (Lines 3–4), and then iterates over the cars in T_o (Lines 6–13). The cars are inserted into the feasible positions of the sequence, determined by Algorithm 6 (Lines 8–11), until $L_o - L_{o-1}$ is reached (Lines 12–13). Finally, the primal solution is returned.

Algorithm 5: GeneratePrimalSolution()

```
1  π ← T_0
2  for o ← 1, ..., |C| do
3  |  if L_o − L_{o−1} = 0 then
4  |  |  return π
5  |  inserted ← 0
6  |  foreach v ∈ T_o do
7  |  |  for t ← 1, ..., |π| do
8  |  |  |  if FeasibleInsertion(π, i | v ∈ Q_i, t) = true then
9  |  |  |  |  Insert car v in position t of π
10 |  |  |  |  inserted ← inserted + 1
11 |  |  |  |  break
12 |  |  if inserted = L_o − L_{o−1} then
13 |  |  |  break
14 return π
```

Algorithm 6: FeasibleInsertion(π, i, t)

foreach $j \in C$ **do**
  if $c_{ji} = 1$ **and** there are more than p_j cars with option j in any window of q_j positions, from position $t - (q_j - 1)$ to position $t + q_j - 2$, when a car of class i is considered to be between π_{t-1} and π_t **then**
    return *false*
return *true*

7 Heuristic algorithm

Preliminary tests showed that the performance of both the proposed formulations and the exact strategies could be improved by using a good primal bound. To this end, we develop a heuristic approach based on ILS. This metaheuristic was also employed in Ribeiro et al. [2008] for the ROADEF'2005 Challenge version of the CSP, specifically for minimizing the violation of priority constraints. The pseudocode of our method is summarized in Algorithm 7. The algorithm is multi-start and it is executed I_R times (Lines 2–18). At each iteration, there is a constructive procedure (Line 3) followed by intensification and diversification steps consisting of local search (Line 9) and perturbation (Line 15) mechanisms, respectively, performed I_{ILS} times (Lines 8–16) without improvement (Lines 10–14). Parameter n is associated with the number of perturbation moves, as later described in Section 7.4.

Algorithm 7: $\text{ILS}_{\text{Max-CSP}}(I_R, I_{ILS}, n)$

```
1  π* ← ∅
2  for iterR ← 1, ..., I_R do
3  |  π ← GenerateInitialSolution()
4  |  if |π| = D then
5  |  |  return π
6  |  π' ← π
7  |  iterILS ← 0
8  |  while iterILS < I_ILS do
9  |  |  π ← Insertion(π)
10 |  |  if |π| > |π'| then
11 |  |  |  π' ← π
12 |  |  |  if |π'| = D then
13 |  |  |  |  return π'
14 |  |  |  IterILS ← 0
15 |  |  π ← Perturbation(π', n)
16 |  |  iterILS ← iterILS + 1
17 |  if |π'| > |π*| then
18 |  |  π* ← π'
19 return π*
```

7.1 Solution representation

The solution is represented by a sequence π of cars. A car of class i in a given position t of the sequence can be written as $\pi_t = i$. For example, if $\pi = [1, 2, 3, 2]$, we have $\pi_1 = 1$, $\pi_2 = 2$, $\pi_3 = 3$, and $\pi_4 = 2$. Moreover, it is possible to concatenate π with another car using the operator $\oplus$, that is, $\pi \oplus 4 = [1, 2, 3, 2, 4]$.

7.2 Constructive procedure

The constructive procedure is presented in Algorithm 8. P' denotes the subset of P containing classes with at least one unscheduled car. P' is therefore initialized to be

equal to P (Line 1), because the initial solution is empty. CP is the list of candidate classes for insertion, and this is also initialized with P' (Line 2). While CP is not empty, the algorithm randomly chooses a class from the list and appends one of its cars at the end of the sequence (Lines 5–6). Next, the list CP is updated with the classes that can still be appended at the end of the sequence after the insertion. More precisely, for each class i present in P', one verifies the following. For every option $j \in C$ present on that class, one checks if the $q_j - 1$ latest scheduled positions of the current solution do not contain more than $p_j - 1$ cars with option j (Lines 14–15). If all options constraints are satisfied, this class joins CP (Lines 18–19).

Algorithm 8: GenerateInitialSolution()

```
1  P' ← P                                  ▷ collection of classes that still have unscheduled cars
2  CP ← P'                                 ▷ list of candidate classes for insertion
3  π ← ∅
4  while CP ≠ ∅ do
5  |  i ← class from CP selected at random
6  |  π ← π ⊕ i                            ▷ concatenate π with car of class i
7  |  CP ← ∅
8  |  foreach i ∈ P' do
9  |  |  if there are d_i cars of class i in π then
10 |  |  |  P' ← P' \ i
11 |  |  |  continue        ▷ move to another class since all cars from P are already in π
12 |  |  feasible ← true
13 |  |  foreach j ∈ C do
14 |  |  |  if c_ji = 1 and there are more than p_j − 1 cars with option j in the last q_j − 1
   |  |  |    positions then
15 |  |  |  |  feasible ← false
16 |  |  |  if feasible = false then
17 |  |  |  |  break
18 |  |  if feasible = true then
19 |  |  |  CP ← CP ∪ i
20 return π
```

7.3 Insertion mechanism

The insertion mechanism is presented in Algorithm 9. It starts at a random position *beginning* (Line 1) and attempts insertion in any position of the sequence, from *beginning* until the end of the sequence, and then before the first car until the position *beginning* (Lines 3–11). For each position t, one checks the feasibility (see Algorithm 6) of inserting cars that belong to the classes in P'. As per Section 7.2, P' is the subset of classes with at least one unscheduled car. For each class $i \in P'$, if the insertion of a car of class i is feasible, it is scheduled in position t (Lines 8–9). Note that *beginning* must be increased by one unit if a car is inserted in a position prior to it (Lines 10–11).

Algorithm 9: Insertion(π)

```
1  beginning ← value at random selected from the set {1, 2, ..., |π|}
2  P' ← P                                  ▷ collection of classes that still have unscheduled cars
3  for t ← beginning, ..., |π|, 1, ..., beginning − 1 do
4  |  foreach i ∈ P' do
5  |  |  if there are d_i cars of class i in π then
6  |  |  |  P' ← P' \ i
7  |  |  |  continue          ▷ move to another class since all cars from i are already in π
8  |  |  if FeasibleInsertion(π, i, t) = true then
9  |  |  |  Insert car of class i in position t ∈ π
10 |  |  |  if t < beginning then
11 |  |  |  |  beginning ← beginning + 1
12 return π
```

7.4 Perturbation mechanisms

We have implemented two perturbation mechanisms, namely, removal (Algorithm 10) and swap (Algorithm 12). The latter is equivalent to the car exchange method presented in Ribeiro et al. [2008], yet applied during the diversification step instead of in the intensification phase since it does not directly affect the value of our objective function but might allow for future insertions. Also, removing a car of class i during perturbation and inserting a car of the same class in the following local search yields a similar behavior to their car insertion neighborhood. The authors emphasized both procedures, car exchange and car insertion, for having an efficient nature of quick evaluation. As demonstrated in Section 8.2, selecting a random perturbation operator outperforms using one of the mechanisms individually. Note that both algorithms receive the parameter n, which determines the maximum number of times a move from the corresponding mechanism is performed per perturbation call. Moreover, it is important to emphasize that our heuristic only considers feasible sequences throughout its execution, meaning that any acceptable change in the solution will always lead to another feasible schedule.

7.4.1 Removal

For the removal operator, we use a list of positions CT in which the car can be removed from the sequence without violating any option constraint. This list is updated after each removal is performed.

Algorithm 10 presents the removal perturbation procedure. First, a constant Δ is defined (Line 1) to enable an efficient update of CT due to a car removal from the solution. Such constant specifies the number of backward/forward consecutive positions from where a car was removed that can be either added to CT or deleted from it. The value of Δ is explained further in detail in Algorithm 11. Next, CT is initialized (Line 2), one of its positions is randomly selected (Line 3) and the car scheduled in this position is

removed from π (Line 4). The loop (Lines 5–8) is responsible for the next $n-1$ removals. Before each of these removals, the interval for updating CT ranges from Δ positions before and $\Delta - 1$ positions after the last removal position (Line 6).

Algorithm 10: Removal(π, n)

```
1 Δ = max_{i∈C}{q_j} − 1                          ▷ the maximum option window of the instance
2 CT ← UpdateCT(π, 2, |π| − 1, ∅)                 ▷ list of candidate positions for removal
3 t ← position from CT selected at random
4 Update π by removing car π_t
5 for iter ← 1, ..., n − 1 do
6 |   CT ← UpdateCT(π, max{t − Δ, 2}, min{t + Δ − 1, |π| − 1},CT)
7 |   t ← position from CT selected at random
8 |   Update π by removing car π_t
9 return π
```

Algorithm 11: UpdateCT($\pi, beginning, end$,CT)

```
1  if CT = ∅ then
2  |   CT ← CT ∪ {1}                              ▷ removing the first car of π is always feasible
3  |   CT ← CT ∪ {|π|}                            ▷ removing the last car of π is always feasible
4  else
5  |   Update CT by decreasing one unit from the value of the positions belonging to CT
   |     that are greater than end                ▷ a car was removed before them
6  for t ← beginning, ..., end do
7  |   feasible ← true
8  |   foreach j ∈ C do
9  |   |   if there are more than p_j cars with option j in any window of q_j positions, from
   |   |     position t − (q_j − 1) to position t + q_j − 2, disregarding t then
10 |   |   |   feasible ← false
11 |   |   |   break
12 |   if feasible = true then
13 |   |   CT ← CT ∪ {t}
14 |   else
15 |   |   CT ← CT \ {t}
16 return CT
```

Algorithm 11 details how CT is updated given a range of positions from *beginning* to *end* that might have been affected by a removal. Note that removing the cars associated with the first and the last positions of the sequence is always feasible. Hence, such positions are directly added to CT when the list is being initialized (Lines 1–3). If CT is not empty and therefore being updated, the candidate positions prior to *beginning* remain the same, and the ones after *end* are decreased by one unit in CT because a car was removed before them (Lines 4–5). Next, the positions in the given range are verified and eventually updated (Lines 6–15). Let t be a position in this range. For all options $j \in C$, the procedure verifies if their associated constraints are not violated when π_t is removed (Lines 8–11). Furthermore, in order to determine the feasibility of removing a car in a given position t, one needs to check the $q_j - 1$ positions before and after t for all $j \in C$ (Line 9). Thus, if then the removal of a car from position t affects the feasibility

of removing cars from positions $k + \Delta \geq t \mid k < t$, and $k - \Delta < t \mid k > t$. This is why the range passed to Algorithm 11 after the first removal starts at $beginning = \max\{t - \Delta, 2\}$ and terminates at $end = \min\{t + \Delta - 1, |\pi| - 1\}$ (Algorithm 10, Line 6).

When CT is implemented as a linked list, the time complexity of the update is $\mathcal{O}(|\mathrm{CT}| + |C|\Delta^2)$, which corresponds to the number of iterations over CT to decrease one unit from the positions after *end* plus the number of calls for checking the feasibility of removing ($\mathcal{O}(|C|\Delta)$) every position inside the 2Δ range. As for initializing CT, we check the feasibility of removing every position in the solution, which leads to a time complexity of $\mathcal{O}(|\pi||C|\Delta)$. It is worth mentioning that a trivial procedure for this perturbation move would reinitialize CT at each new removal, thus yielding an inferior efficiency compared to Algorithm 10.

7.4.2 Swap

For the swap mechanism, a candidate list is somewhat computationally inefficient because it involves pairs of positions. Instead, a position t_1 from the solution is randomly selected so one can attempt to apply a move involving the cars at t_1 and at any of the remaining positions.

Algorithm 12 presents the swap perturbation procedure. Initially, an array of size $|\pi|$ filled with $false$ values is defined as $infeasPos$ (Line 1), which will store the positions verified to be infeasible for any swap (i.e., if $infeasPos_t = true$, car π_t cannot be swapped with any other car and result in a feasible sequence). After each swap is performed, all values of this array are reset to $false$ since the change in the solution could lead to new swap possibilities including any car. An efficient update for $infeasPos$ uses a list, here denoted as $truePos$, containing the positions that have been set to $true$ during the perturbation. This list improves the time complexity of resetting $infeasPos$ from $\mathcal{O}(|\pi|)$ to $\mathcal{O}(n)$, considering that the maximum size of $truePos$ is n.

At first, $truePos$ is initialized empty (Line 2), and for each iteration (Lines 3–15), one randomly selects a position t_1 from the solution (Line 4), which needs to satisfy $infeasPos_{t_1} = false$ (Lines 5–6). Position t_1 is then marked as already selected (Lines 7–8), and the search for a car to swap with π_{t_1} begins from another random position t_2 of the solution. If $infeasPos_{t_2} = false$ and t_2 contains a car that can be swapped with π_{t_1}, the move is performed and all values of $infeasPos$ are reset to false (Lines 11–15). If no feasible swap exists involving t_1, the algorithm moves on to the next iteration with $infeasPos_{t_1} = true$. Note that, because there can be an iteration without any swap, at most n swaps are performed.

Algorithm 13 describes the steps for determining whether it is feasible or not to swap the cars scheduled in the positions t_1 and t_2. It is necessary to iterate over the

Algorithm 12: Swap(π, n)

```
infeasPos ← Boolean array of size |π| filled with false
truePos ← ∅
for iter ← 1, ..., n do
    t1 ← position from π selected at random
    while infeasPos_{t1} = true do
        t1 ← (t1 + 1) mod |π|                          ▷ circular increment inside π
    infeasPos_{t1} ← true
    truePos ← truePos ∪ t1
    beginning ← position from π selected at random
    for t2 ← beginning, ..., |π|, 1, ..., beginning − 1 do
        if infeasPos_{t2} = false and FeasibleSwap(π, t1, t2) = true then
            Update π by swapping cars π_{t1} and π_{t2}
            Update infeasPos by resetting the value false to the positions in truePos
            truePos ← ∅
            break
return π
```

Algorithm 13: FeasibleSwap(π, t_1, t_2)

```
if π_{t1} = π_{t2} then
    return false
foreach j ∈ C do
    if c_{jπ_{t1}} = 1 ≠ c_{jπ_{t2}} and there are more than p_j cars with option j in any window of q_j
      positions, from position t2 − (q_j − 1) to position t2 + q_j − 1, when π_{t1} is considered to
      be at position t2 then
        return false
    else if c_{jπ_{t2}} = 1 ≠ c_{jπ_{t1}} and there are more than p_j cars with option j in any window
      of q_j positions, from position t1 − (q_j − 1) to position t1 + q_j − 1, when π_{t2} is
      considered to be at position t1 then
        return false
return true
```

options $j \in C$ to check the move feasibility (Lines 3–7). However, cars that belong to the same class should not be swapped, because it would lead to an equivalent solution, so the algorithm returns $false$ immediately if this is the case (Lines 1–2). For cars that belong to different classes, if an option is shared by both cars or neither requires it, the constraint will not be affected by the swap. The feasibility of placing the car scheduled at position t_1 at position t_2 is proven false if any of the conditions stated in Lines 4 and 6 are satisfied. Lastly, the procedure returns *true* if there is no violation of the option constraints (Line 8).

8 Computational experiments

All formulations and algorithms were coded in C++ and executed on an Intel Xeon E5-2650 Processor with 128 GB of RAM memory running Ubuntu Linux 16.04. CPLEX 22.1 was adopted as the MIP solver with the strong branching variable selection strategy. We set a time limit of 600 seconds for each instance.

8.1 Instances

In our experiments, we categorized the instances into two sets: the literature instances (Set 1) and the real demands obtained from our partner company (Set 2). The selection of the 247 instances in Set 1 was done by Sun et al. [2023] and is publicly available at `https://github.com/yuansuny/CSP/tree/main/testset`. As for Set 2, they consist of four months of historical data from the company that motivated this study. Each instance is named based on the corresponding month, the index of the shift (sorted chronologically), and the number of cars. It is worth noting that all options in these 165 instances follow either of the formats $(p_j = 1, q_j)$ or $(p_j = q_j - 1, q_j)$. We would like to note there are certain instances in Set 2 where the number of cars deviates significantly from the norm, for example, $D = 1$. We chose not to exclude these instances from the dataset because they can actually make the monthly simulation more challenging (i.e., more shift delays to assemble a car). This is because it is more likely that cars unscheduled from previous shifts would not be able to fit into a schedule with a low demand of $D = 1$ due to the spacing constraints imposed by the options. In other words, there might not be a sufficient number of cars available on that given shift to respect the required distancing. All instances used in our study are available at `https://github.com/laradicp/max-csp/tree/main/instances`.

8.2 Parameter tuning

Concerning the number n of moves performed in a single perturbation, the type of perturbation mechanism, and the main parameters I_R and I_{ILS}, we ran several tests in order to choose a good compromise between solution quality and CPU time.

Figure 5 introduces three different perturbation mechanisms alongside the tuning of parameter n. The variations of the perturbation procedure are removal (using only Algorithm 10), swap (using only Algorithm 12), and both (selecting randomly which algorithm to run at each perturbation call). Given the current solution π, the evaluated settings for parameter n were $\min\{|\pi|/2, D/50\}$, $\min\{|\pi|/2, D/25\}$, $\min\{|\pi|/2, D/20\}$, $\min\{|\pi|/2, D/15\}$, $\min\{|\pi|/2, D/10\}$ and $\min\{|\pi|/2, D/5\}$. We ran Algorithm 7 ten times on each instance within the subset of instances from Set 1 that remained unsolved

by both formulations F1 and F2, considering $I_R = 1$ and $I_{ILS} = D/2$, and computed the corresponding average gap and average CPU time of these executions. The graph shows that the smallest gap, with still very short average CPU time, was obtained by Setting 18 ($n = \min\{|\pi|/2, D/5\}$). Thus, this was the chosen setting.

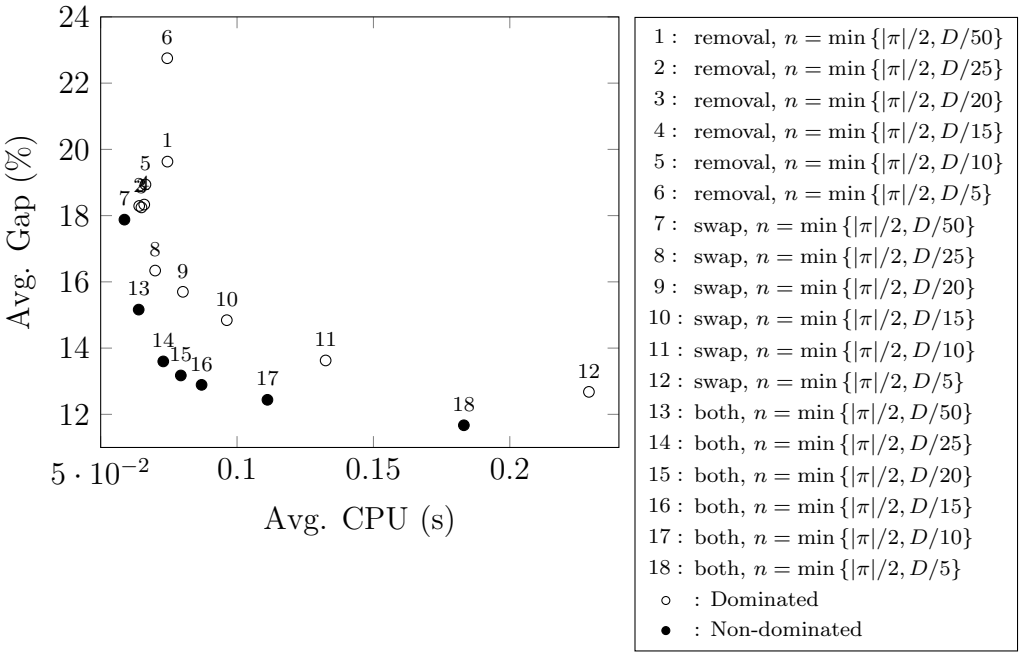

Figure 5: Impact of perturbation mechanisms and tuning of parameter n.

Next, we discuss Figure 6 in regard to the main parameters of the heuristic algorithm: I_R and I_{ILS}. The values considered for I_R were 5, 10, 15, 25 and 50, while those for I_{ILS} were $D/2$, D and $2 \times D$. We ran the algorithm ten times on the same subset of instances from Set 1 unsolved by both F1 and F2 models. According to the graph, Setting 12 has a relevant gap difference when compared to Setting 9, yet they are still close enough in average CPU time. Setting 15, on the other hand, requires double the time of Setting 12. Therefore, we decided to adopt Setting 12, with $I_R = 25$ and $I_{ILS} = 2 \times D$.

8.3 Results

In the following, we first discuss the performance of the bounds presented in Section 6 and of the heuristic solutions presented in Section 7. We then test the performance of the exact algorithms on the instances in Set 1, which are typically more challenging than those in Set 2 and can be better used to evaluate the different methods.

We compare the combinatorial UB described in Section 6.1 with the root node relaxation of formulation F1 given by CPLEX. When it comes to the instances in Set 2, we found an average percentage gap of 0.32% with respect to the best-known primal solution for both the root node relaxation of CPLEX and the combinatorial UB, but the

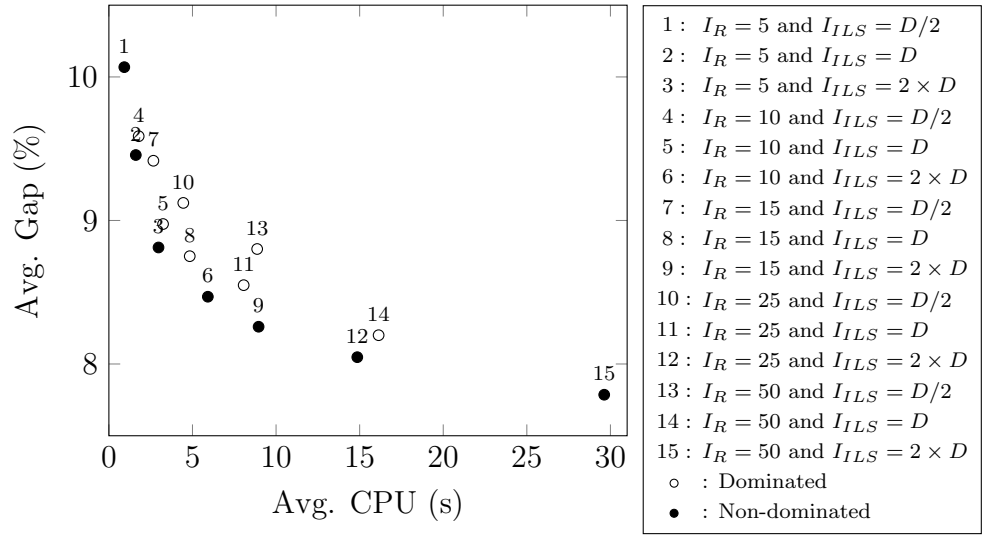

Figure 6: Parameter tuning: I_R and I_{ILS}.

CPU time of the former is on average 122.85% longer than the latter. As for the instances in Set 1, however, the combinatorial UB yields a trivial result (i.e., the total number of cars to be scheduled), thus suggesting that the bound does not prove effective in this instance set. Detailed results are available at `https://github.com/laradicp/max-csp/tree/main/results`.

For the LBs, we compare the performance of the combinatorial LB described in Section 6.2 with the heuristic bound described in Section 7. This information is summarized in Table 1, from which it becomes clear that the heuristic method is crucial for attaining good primal solutions.

Table 1: Comparison between combinatorial and heuristic LBs in each instance set.

instance set	bound	avg. gap (%)	avg. time (s)
Set 1	combinatorial	3829.34	< 0.01
	heuristic	5.13	5.52
Set 2	combinatorial	34.22	< 0.01
	heuristic	0.04	0.13

We present the results of the computational experiments for Set 1 in two tables: Table 2 focuses on the performance of the exact methods without the heuristic initialization, and Table 3 with the heuristic initialization. The associated bound is used for formulations I_I, I_D, and I_B, and the primal solution itself is given to CPLEX as a MIP start for formulations $F1$, $F2$, $F2_I$, $F2_D$, and $F2_B$. Both tables provide the following information: in "optimal (%)", we have the percentage of known optimal solutions found; in "smallest (%)", the percentage of instances in which the gap was the smallest; in "gap (%)", the average gap based on the best dual value (as in Equation (29)); in "time (s)", the average CPU time in seconds, considering all instances; and in "time* (s)", the average

CPU time in seconds, not considering the instances that exceeded the time limit.

$$gap_{primal}(\%) = \frac{bestDual - primal}{primal} \times 100 \tag{29}$$

Table 2: Performance of different uninitialized methods on the instances in Set 1.

method	avg. gap (%)	optimal (%)	smallest (%)	avg. time (s)	avg. time* (s)
F1	65.00	58.30	67.61	288.94	66.43
I_I	4.52	40.89	44.94	405.88	120.47
I_D	2300.65	58.30	58.30	290.31	68.76
I_B	3.43	49.39	53.85	350.47	90.60
F2	240.01	57.89	65.18	305.95	92.08
$F2_I$	53.28	55.06	62.75	311.11	75.32
$F2_D$	53.39	55.06	62.75	311.20	75.46
$F2_B$	53.33	55.06	62.75	311.14	75.37

Table 3: Performance of different initialized methods on the instances in Set 1.

method	avg. gap (%)	optimal (%)	smallest (%)	avg. time (s)	avg. time* (s)
F1	2.08	59.51	71.26	285.47	71.51
I_I	2.53	46.96	57.09	364.03	97.50
I_D	3.42	57.09	60.73	290.94	58.57
I_B	2.54	51.82	59.92	337.16	92.78
F2	2.34	58.70	68.42	292.08	75.45
$F2_I$	2.46	53.44	60.32	310.79	58.81
$F2_D$	2.46	53.44	60.32	310.80	58.83
$F2_B$	2.38	53.85	62.35	308.77	59.13

Regarding the uninitialized methods, F1, F2, and I_D demonstrate strength in finding relatively high numbers of optimal solutions. However, a notable weakness is their inconsistency due to the high gaps observed, indicating that the suboptimal solution is poor when the optimal is not found. For example, each iteration of I_D can only return an infeasible, a time limit, or an optimal status. In the first two cases, it uses the combinatorial primal solution, which is very low-performing for the instances in Set 1, according to Table 1. On the other hand, methods I_I and I_B exhibit significant strengths, particularly in achieving lower optimality gaps, especially I_B. This suggests it may be the preferred choice when a good primal bound is unavailable to warm-start CPLEX. These methods consistently produce good quality solutions, enhancing their reliability in the industry scenario.

With the heuristic initialization, the weaknesses previously shown by F1, F2, and I_D are overcome. F1 dominates all the other methods in performance, consistently finding optimal solutions and achieving the smallest gaps. Importantly, it achieves these strengths while maintaining efficient CPU times.

In both tables, I_I, I_D, and F2 are dominated by another method. Thus, they are not the most competitive options in terms of solution optimality and computational efficiency. Moreover, $F2_I$, $F2_D$, and $F2_B$ performed similarly, suggesting that assigning

different weights to the variables in SOS1 does not stand out as the most critical strategy for the problem at hand.

8.4 Instance space analysis

Sun et al. [2023] previously performed an *instance space analysis* (ISA) on the CSP. An ISA is capable of generating a plot of instances based on the projection of various features into a 2D space. Since our approach introduces a novel objective function, and we propose new algorithms tailored to this specific context, a fresh examination of the instance space is necessary to provide a comprehensive evaluation of our variant. We conducted an ISA for the Max-CSP by using the *Melbourne algorithm test instance library with data analytics* (MATILDA) framework [Smith-Miles et al., 2020].

MATILDA was utilized with the following settings: minimization optimization criteria of the cost measure (here gap); absolute performance criteria of 0.0% (i.e., reaching proven optimality); correlation threshold for feature selection of 0.4; and default settings for other parameters. We included all uninitialized and initialized algorithms in the input data. Additionally, we incorporated the 14 instance features defined by Sun et al. [2023] into our analysis. From this set, seven were selected by MATILDA for the projection.

It may be necessary to analyze multiple graphs simultaneously in order to obtain meaningful conclusions when interpreting an ISA. For example, Figure 7 shows that the heuristic algorithm performs worse for the instances projected in the lower right half of the 2D space, as optimality was not reached in that section.

Concurrently, Figure 8 exhibits the value of the maximum of option utilization (max-utilization) feature in each instance. By comparing both graphs, we can infer a positive correlation tendency between this feature and the instance difficulty. In other words, the higher the maximum of option utilization, the harder the instance tends to be for the heuristic algorithm to solve.

Similarly, Figure 9 suggests a positive correlation tendency between the instance difficulty and the average of option utilization (ave-utilization), and negative correlation tendencies with respect to the instance difficulty can be observed with the minimum of p/q ratio (min-pq-ratio) in Figure 10 and the average of p/q ratio (ave-pq-ratio) in Figure 11.

The numbers Z_1 and Z_2 are the coordinates of each instance in the 2D projection, calculated by multiplying the weights obtained by the analysis with the values of the features in the respective instance. We provide the weights used in our ISA, defined by MATILDA, in case one wants to plot a new instance in this same space.

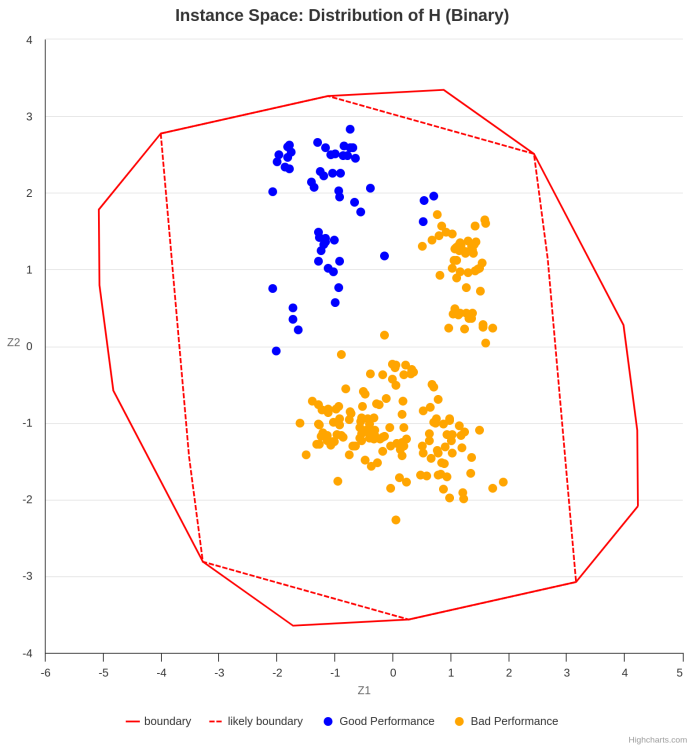

Figure 7: Binary distribution of heuristic performance on the instances in Set 1.

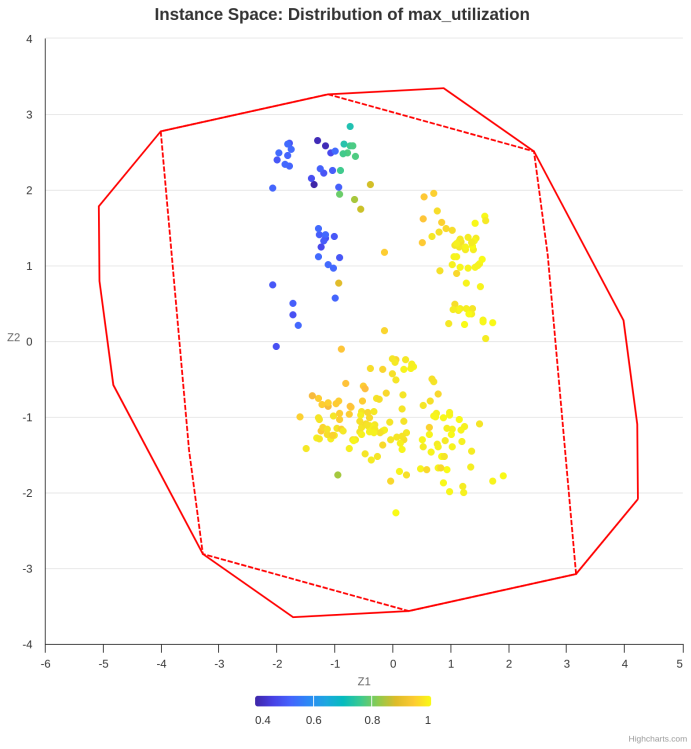

Figure 8: Distribution of the maximum of option utilization in the instances in Set 1.

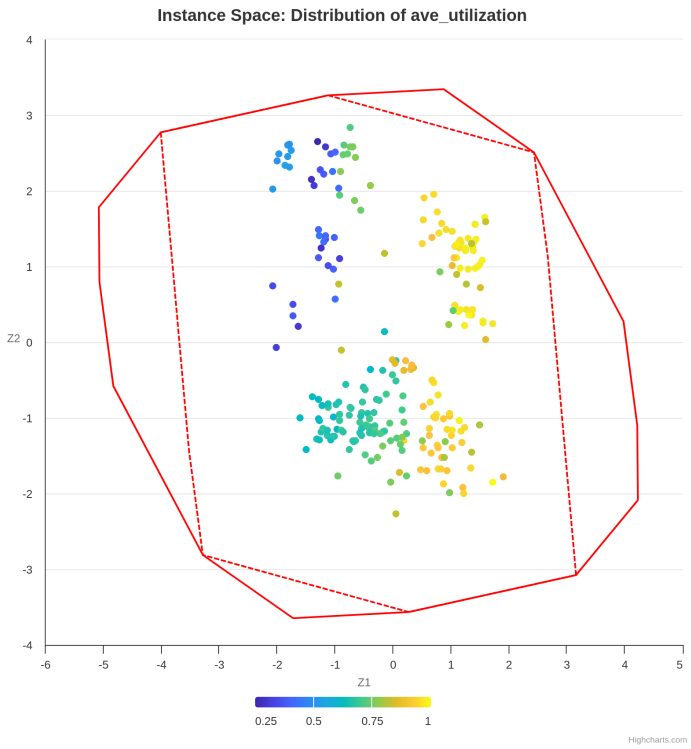

Figure 9: Distribution of the average of option utilization in the instances in Set 1.

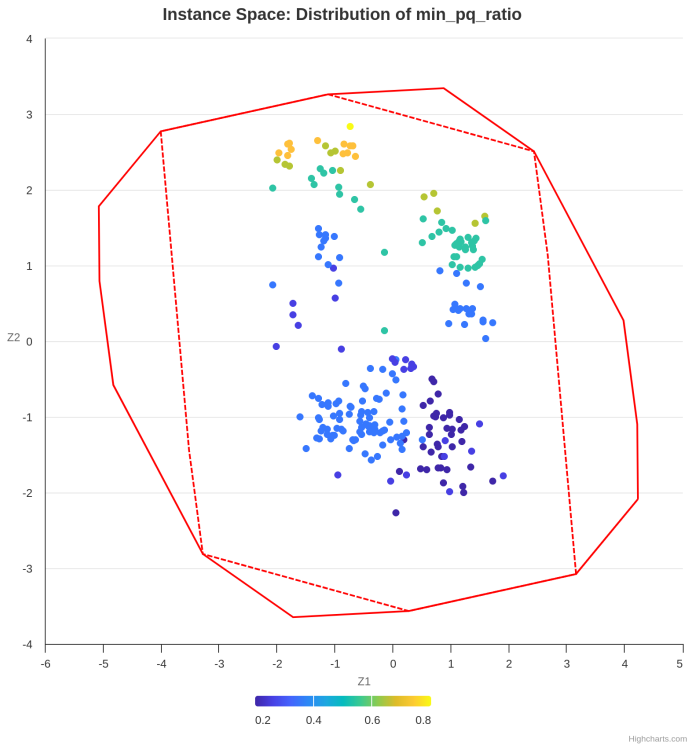

Figure 10: Distribution of the minimum of p/q ratio in the instances in Set 1.

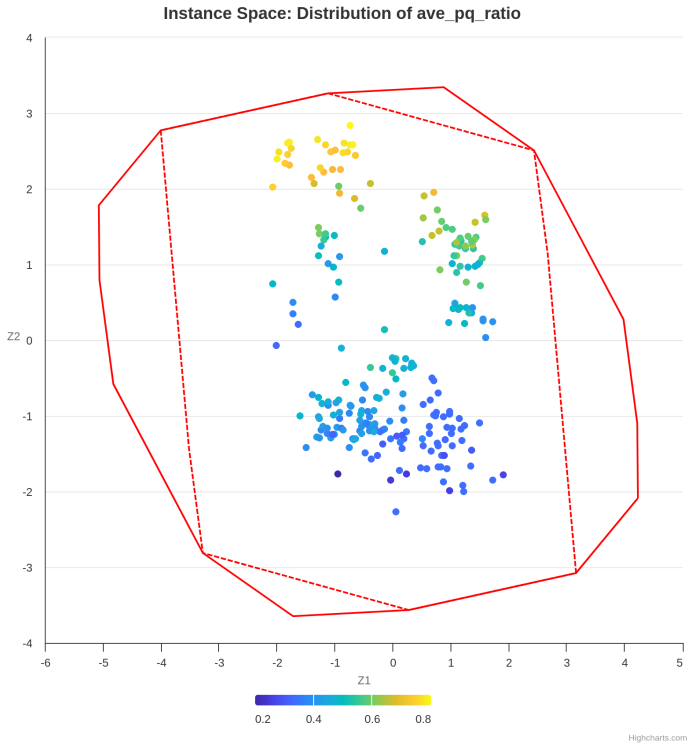

Figure 11: Distribution of the average of p/q ratio in the instances in Set 1.

$$\begin{bmatrix} Z_1 \\ Z_2 \end{bmatrix} = \begin{bmatrix} -0.081 & 0.4673 \\ 0.5512 & 0.0222 \\ 0.4987 & -0.2672 \\ 0.561 & 0.0951 \\ -0.0032 & 0.2726 \\ -0.3139 & 0.4534 \\ -0.2575 & -0.2387 \end{bmatrix}^T \begin{bmatrix} num-classes \\ ave-utilization \\ max-utilization \\ ave-opt-class \\ min-pq-ratio \\ ave-pq-ratio \\ std-pq-ratio \end{bmatrix}$$

Please note that detailed results and graphs are provided in the appendix.

8.5 Real-world instances and operational insights

Having primarily focused on the instances in Set 1, which have demonstrated to be more challenging, we now analyze the computational results of the instances in Set 2, compare the performance of each method on them, and simulate the cumulative sequencing within the monthly time horizon. The last shift of each month is solved by the Slow-CSP formulation defined in Section 4.

First, we present Tables 4 and 5, which are similar to Tables 2 and 3, but without the columns "optimal (%)"and "smallest (%)"since every method yielded an optimal solution. This comparison reveals that all formulations exhibit efficient performance on the instances in Set 2, with the exception of F1 and the uninitialized I_I. The latter is an unexpected behavior because warm starting typically enhances performance; however, it

appears to have the opposite effect for F1 in this particular context.

Table 4: Performance of different uninitialized methods on the instances in Set 2.

method	avg. gap (%)	avg. time (s)	avg. time* (s)
F1	0.00	5.25	5.25
I_I	0.00	3.93	3.93
I_D	0.00	0.29	0.29
I_B	0.00	0.73	0.73
F2	0.00	0.44	0.44
$F2_I$	0.00	0.40	0.40
$F2_D$	0.00	0.40	0.40
$F2_B$	0.00	0.39	0.39

Table 5: Performance of different initialized methods on the instances in Set 2.

method	avg. gap (%)	avg. time (s)	avg. time* (s)
F1	0.00	6.43	2.81
I_I	0.00	0.40	0.40
I_D	0.00	0.29	0.29
I_B	0.00	0.22	0.22
F2	0.00	0.48	0.48
$F2_I$	0.00	0.45	0.45
$F2_D$	0.00	0.45	0.45
$F2_B$	0.00	0.43	0.43

Next, we discuss the monthly simulation. The dataset covers demands for four months: January, February, March, and May of 2019. It is important to note that the number of cars in different classes is not spread evenly across these months. All cars were produced by the end of January, February, and May. However, in March, a total of 240 cars were left unscheduled for the dedicated shift. According to the Slow-CSP solution, which was obtained in 0.09 seconds of CPU time, these cars could be sequenced in a single shift if the assembly line moved at half of the usual speed. In comparison, if the objective function by Bautista et al. [2008] was applied (considering unitary weights for the "upper over-assignment" and zero weights for the "upper under-assignment"), a total of 248 option constraint violations would be found within the same month of production, leading to many delays and disruptions.

Hence, the Set 2 instances seem within the reach of our methodologies and can be solved to optimality in less than a second. Our simulation over a few months of data confirms that the operational choice of rolling unscheduled cars to the next shift does not generate bottlenecks. The unscheduled cars left at the end of the production horizon can be easily accommodated in a single shift, and this latter problem can be solved by means of the proposed Slow-CSP.

9 Concluding remarks

In this paper, we investigated a newly proposed variant of the CSP, denoted as Max-CSP, which maximizes the number of cars sequenced in the assembly line without violating any option constraints. This different objective function was brought by a multinational automotive partner company, in the context of their Brazilian assembly line, with the particularity of having only "hard sequencing rules", that is, any option constraint violation leads to an interruption in the assembly line. The full interruption is the least desirable outcome for the company. Thus, it is preferable that cars are left unscheduled if they would cause a violation of the option constraints. The unscheduled cars are then considered in the demand of the subsequent shift until they are included in the sequencing. At the end of a monthly time horizon, the company runs a dedicated line to schedule all remaining cars. This shift is operated at a slower pace in order to proportionally loosen the option constraints and allow all cars to be produced. In that context, it is crucial for the company that we provide the maximum pace of this line, for which purpose we develop a secondary model, called the Slow-CSP.

We addressed the Max-CSP from different perspectives, more precisely, we devised: (i) an ILP model; (ii) exact algorithms over a feasibility checking ILP model based on iterative and binary searches; (iii) valid combinatorial lower and upper bounds; and (iv) an ILS-based heuristic. Extensive computational experiments were carried out on instances available in the literature. When solely relying on the combinatorial bounds to initialize the exact algorithms, our findings suggest that the incremental iterative approach and the binary search yielded a more consistent performance than the other settings, including the standalone ILP model. Nevertheless, when initializing the exact methods with the high-quality primal bounds achieved by ILS, the ILP model seemed to outperform the other procedures in most cases. Furthermore, we conducted an ISA to identify the features that are likely to affect the performance of the different methods. Finally, we simulated the cumulative sequencing within the monthly time horizon by using real-world demands, divided into shifts. We considered data from four months, and we employed the ILP model for the Max-CSP in all shifts but the last one from every month, whereas the mixed ILP model developed for the Slow-CSP was utilized to schedule the cars that were left out after solving the first model. The results achieved were very promising, as it was only necessary to solve the Slow-CSP in one of the four months.

Further research might address more complex problem features. The problem could be extended to parallel assembly lines, in which the set of cars to be scheduled has to be assigned to one of the assembly lines. Other objective functions, such as the one introduced by Souza et al. [2023], could be compared and contrasted. The weight of a car may represent, other than the number of options, its associated revenue or priority.

Bibliography

J. Bautista, J. Pereira, and B. Adenso-Dìaz. A beam search approach for the optimization version of the car sequencing problem. *Annals of Operations Research*, 159:233–244, 2008.

M.L. Fisher and C.D. Ittner. The impact of product variety on automobile assembly operations: Empirical evidence and simulation analysis. *Management Science*, 45:771–786, 1999.

J. Bukchin and M. Masin. Multi-objective design of team oriented assembly systems. *European Journal of Operational Research*, 156(2):326–352, 2004.

S.K. Bhattacharyya, R. Roy, and M.J. Low. A computer simulation system for the evaluation of man assignments on car assembly tracks. *Simulation*, 61:124–133, 1993.

L. Zeltzer, E.-L. Aghezzaf, and V. Limère. Workload balancing and manufacturing complexity levelling in mixed-model assembly lines. *International Journal of Production Research*, 55:2829–2844, 2017.

A.J. McClellan, W.J. Albert, S.L. Fischer, F.A. Seaman, and J.P. Callaghan. Shoulder loading while performing automotive parts assembly tasks: A field study. *Occupational Ergonomics*, 8:81–90, 2009.

B.D. Parrello, W.C. Kabat, and L. Wos. Job-shop scheduling using automated reasoning: A case study of the car-sequencing problem. *Journal of Automated Reasoning*, 2:1–42, 1986.

M. Dincbas, H. Simonis, and P. Van Hentenryck. Solving the car-sequencing problem in constraint logic programming. In *Proceedings of the 8th European Conference on Artificial Intelligence*, ECAI'88, pages 290–295, USA, 1988. Pitman Publishing, Inc.

N. Boysen, M. Fliedner, and A. Scholl. Sequencing mixed-model assembly lines: Survey, classification and model critique. *European Journal of Operational Research*, 192(2): 349–373, 2009.

Tamás Kis. On the complexity of the car sequencing problem. *Operations Research Letters*, 32(4):331–335, 2004.

B. Estellon and F. Gardi. Car sequencing is np-hard: a short proof. *Journal of the Operational Research Society*, 64(10):1503–1504, 2013.

J. Gottlieb, M. Puchta, and C. Solnon. A study of greedy, local search, and ant colony optimization approaches for car sequencing problems. In S. Cagnoni, C. G. Johnson,

J. J. R. Cardalda, E. Marchiori, D. W. Corne, J.-A. Meyer, J. Gottlieb, M. Middendorf, A. Guillot, G. R. Raidl, and E. Hart, editors, *Applications of Evolutionary Computing*, pages 246–257, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

M. Gravel, C. Gagné, and W. L. Price. Review and comparison of three methods for the solution of the car sequencing problem. *Journal of the Operational Research Society*, 56:1287–1295, 2005.

Uli Golle, Franz Rothlauf, and Nils Boysen. Iterative beam search for car sequencing. *Annals of Operations Research*, 226(1):239–254, Mar 2015.

I. Ozan Yilmazlar and Mary E. Kurz. Adaptive local search algorithm for solving car sequencing problem. *Journal of Manufacturing Systems*, 68:635–643, 2023. ISSN 0278-6125.

N. Boysen and M. Fliedner. Comments on "solving real car sequencing problems with ant colony optimization". *European Journal of Operational Research*, 182(1):466–468, 2007. ISSN 0377-2217.

M. Fliedner and N. Boysen. Solving the car sequencing problem via Branch & Bound. *European Journal of Operational Research*, 191(3):1023–1042, 2008.

A. Bolat and C.A. Yano. Scheduling algorithms to minimize utility work at a single station on a paced assembly line. *Production Planning & Control*, 3(4):393–405, 1992.

Andreas Hottenrott, Leon Waidner, and Martin Grunow. Robust car sequencing for automotive assembly. *European Journal of Operational Research*, 291(3):983–994, 2021. ISSN 0377-2217.

B. Smith. Suceed-first or fail-first: A case study in variable and value ordering. In *Third International Conference on the Practical Application of Constraint Technology*, pages 321–330, London, 1997.

K.S. Hindi and G. Ploszajski. Formulation and solution of a selection and sequencing problem in car manufacture. *Computers & Industrial Engineering*, 26(1):203–211, 1994.

L. Perron and P. Shaw. Combining forces to solve the car sequencing problem. In J.-C. Régin and M. Rueher, editors, *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, pages 225–239, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

C. Solnon, V.-D. Cung, A. Nguyen, and C. Artigues. The car sequencing problem: Overview of state-of-the-art methods and industrial case-study of the roadef'2005 challenge problem. *European Journal of Operational Research*, 191(3):912–927, 2008.

Jiaxi Wu, Yongkang Ding, and Leyuan Shi. Mathematical modeling and heuristic approaches for a multi-stage car sequencing problem. *Computers & Industrial Engineering*, 152:107008, 2021. ISSN 0360-8352.

Haida Zhang and Wensi Ding. A decomposition algorithm for dynamic car sequencing problems with buffers. *Applied Sciences*, 13(12), 2023. ISSN 2076-3417.

D. R. Thiruvady, B. Meyer, and A. Ernst. Car sequencing with constraint-based aco. In *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation*, GECCO '11, pages 163–170, New York, NY, USA, 2011. Association for Computing Machinery.

D. Thiruvady, A. T. Ernst, and M. Wallace. A Lagrangian-ACO matheuristic for car sequencing. *EURO Journal on Computational Optimization*, 2(4):279–296, 2014.

D. Thiruvady, K. Morgan, A. Amir, and A. T. Ernst. Large neighbourhood search based on mixed integer programming and ant colony optimisation for car sequencing. *International Journal of Production Research*, 58(9):2696–2711, 2020.

Filipe Souza, Diarmuid Grimes, and Barry O'Sullivan. Variable-relationship guided lns for the car sequencing problem. In Luca Longo and Ruairi O'Reilly, editors, *Artificial Intelligence and Cognitive Science*, pages 437–449, Cham, 2023. Springer Nature Switzerland.

M. Yavuz and H. Ergin. Advanced constraint propagation for the combined car sequencing and level scheduling problem. *Computers & Operations Research*, 100:128–139, 2018.

Andreas Drexl, Alf Kimms, and Lars Matthießen. Algorithms for the car sequencing and the level scheduling problem. *Journal of Scheduling*, 9(2):153–176, Apr 2006. ISSN 1099-1425.

Christian Artigues, Emmanuel Hebrard, Valentin Mayer-Eichberger, Mohamed Siala, and Toby Walsh. Sat and hybrid models of the car sequencing problem. In Helmut Simonis, editor, *Integration of AI and OR Techniques in Constraint Programming*, pages 268–283, Cham, 2014. Springer International Publishing. ISBN 978-3-319-07046-9.

Mohamed Siala, Emmanuel Hebrard, and Marie-José Huguet. A study of constraint programming heuristics for the car-sequencing problem. *Engineering Applications of Artificial Intelligence*, 38:34–44, 2015. ISSN 0952-1976.

Terry Warwick and Edward P. K. Tsang. Tackling car sequencing problems using a generic genetic algorithm. *Evolutionary Computation*, 3(3):267–298, 1995.

B. Estellon, F. Gardi, and K. Nouioua. Two local search approaches for solving real-life car sequencing problems. *European Journal of Operational Research*, 191(3):928–944, 2008. ISSN 0377-2217.

Celso C. Ribeiro, Daniel Aloise, Thiago F. Noronha, Caroline Rocha, and Sebastián Urrutia. A hybrid heuristic for a multi-objective real-life car sequencing problem with painting and assembly line constraints. *European Journal of Operational Research*, 191 (3):981–992, 2008. ISSN 0377-2217.

F. Guerre-Chaley, Y. Frein, and R. Bouffard-Vercelli. An efficient procedure for solving a car sequencing problem. In *Proceedings 1995 INRIA/IEEE Symposium on Emerging Technologies and Factory Automation. ETFA'95*, volume 2, pages 385–393 vol.2, 1995.

Ignacio Moya, Manuel Chica, and Joaquín Bautista. Constructive metaheuristics for solving the car sequencing problem under uncertain partial demand. *Computers & Industrial Engineering*, 137:106048, 2019. ISSN 0360-8352.

Y. Sun, S. Esler, D. Thiruvady, A. T. Ernst, X. Li, and K. Morgan. Instance space analysis for the car sequencing problem. *Annals of Operations Research*, page To appear, 2023.

J.M.P. Silva, A. Subramanian, and E. Uchoa. On time-indexed formulations for the parallel machine scheduling problem with a common server. *Engineering Optimization*, 0(0):1–18, 2023.

K. Smith-Miles, M.A. Muñoz, and Neelofar. Melbourne algorithm test instance library with data analytics (matilda). Available at `https://matilda.unimelb.edu.au`, 2020.

Appendices

A Instance space analysis

This appendix provides visual representations of the ISA conducted during the course of this study, generated by the MATILDA framework [Smith-Miles et al., 2020]. The primary goal is to offer a comprehensive view of how some features associated with the instances can make the Max-CSP harder to solve.

Since the problem contains a diverse set of benchmark instances (here Set 1) in which an ISA was already performed by Sun et al. [2023], we focus our analysis on those, rather than on the real demands obtained from our partner company (Set 2).

The first set of graphs, Figures 12–28, illustrates the distribution of good and bad results in the proposed algorithms. The threshold that defines a good result is reaching proven optimality (gap 0.0%) within the time limit of ten minutes. Understanding this binary distribution is crucial as it sheds light on the difficulty of different clusters of instances. In that sense, it is noticeable that the lower right half of the two-dimensional space has typically higher gaps for all algorithms, which suggests a more challenging set of instances.

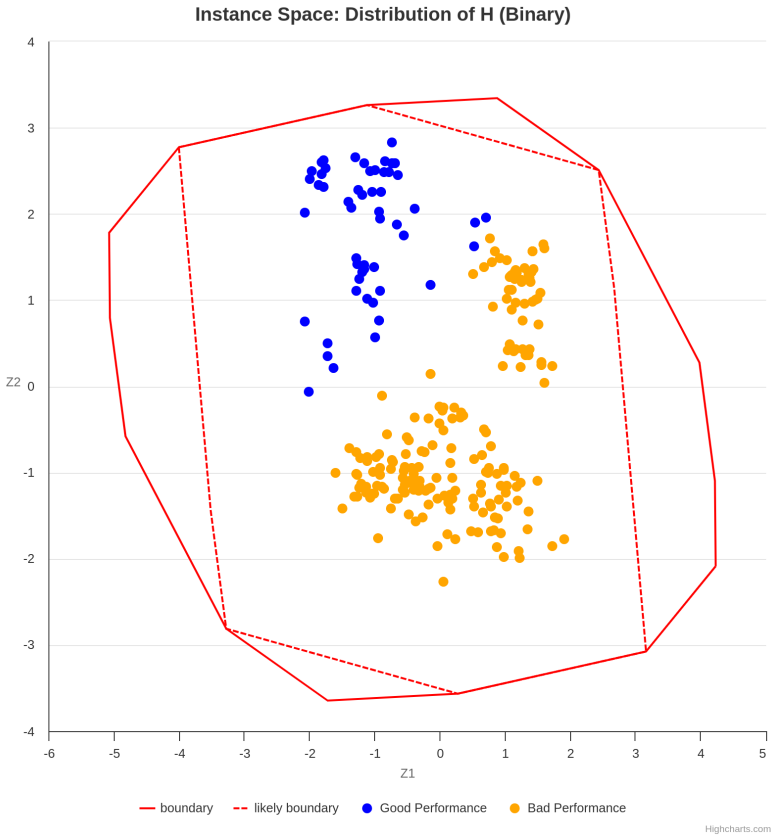

Figure 12: Binary distribution of heuristic performance on the instances in Set 1.

Following this, the set of graphs in Figures 29–35 showcases the distribution of features within the same projection. These visualizations were instrumental in deriving

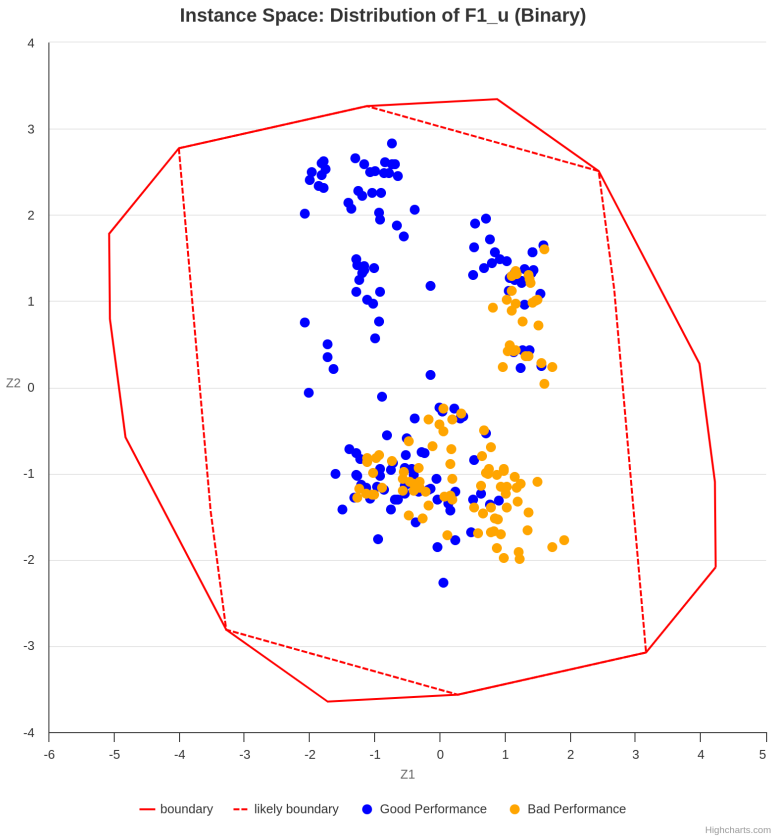

Figure 13: Binary distribution of uninitialized F1 performance on the instances in Set 1.

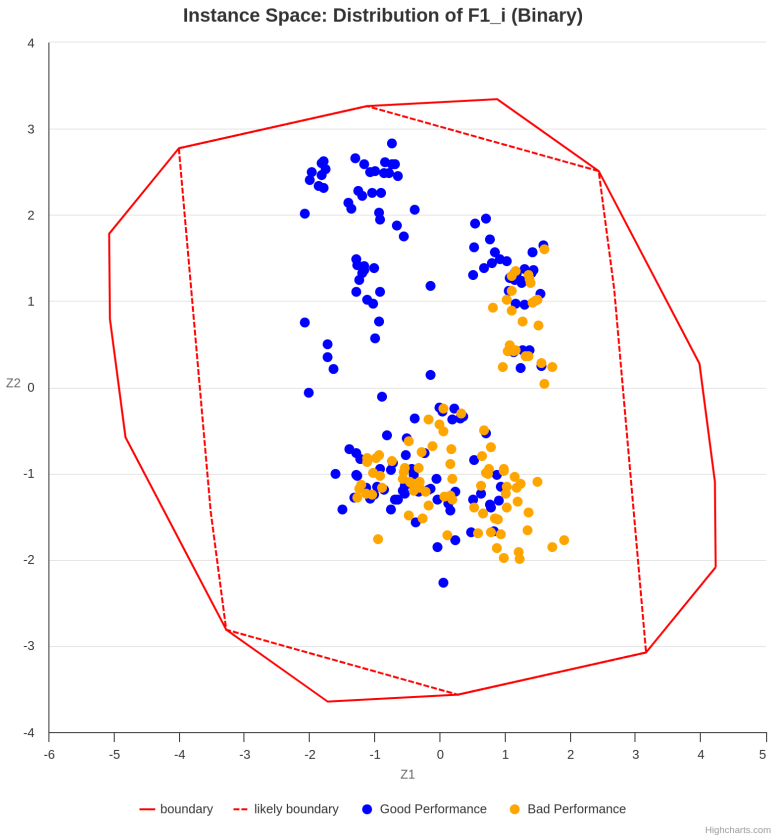

Figure 14: Binary distribution of initialized F1 performance on the instances in Set 1.

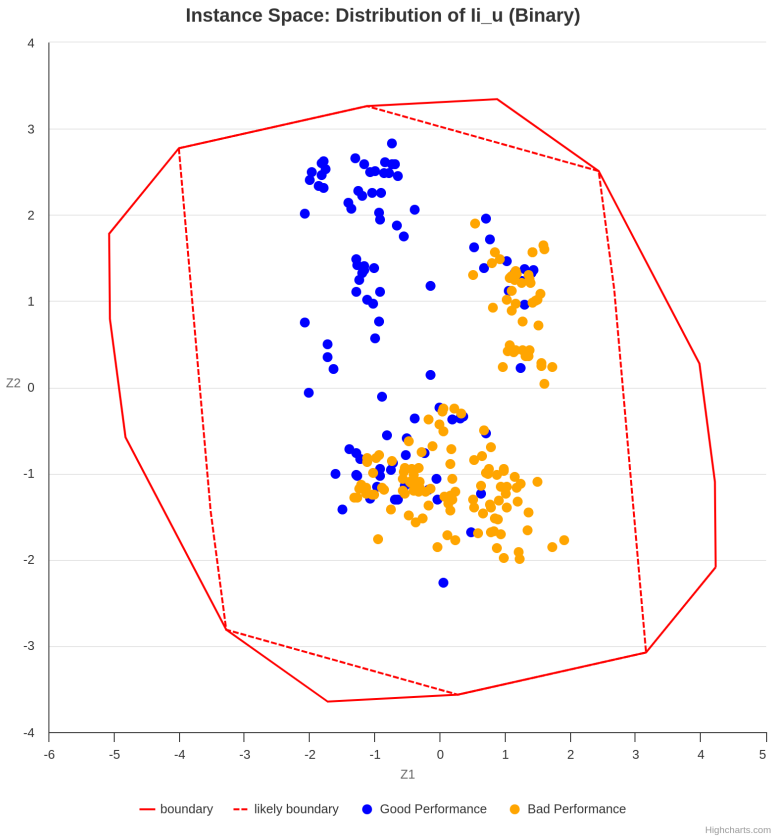

Figure 15: Binary distribution of uninitialized I_I performance on the instances in Set 1.

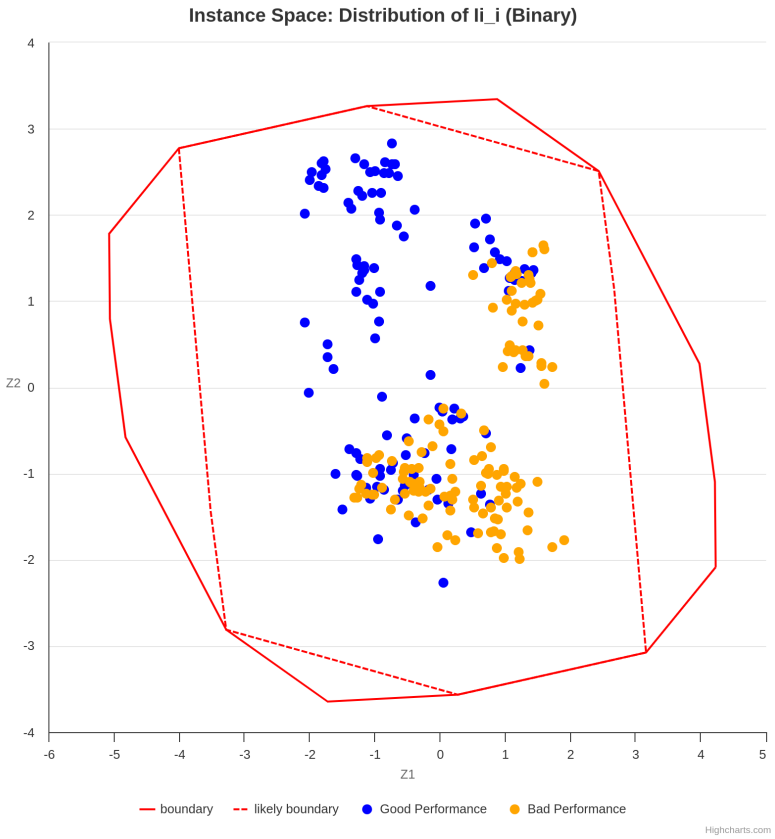

Figure 16: Binary distribution of initialized I_I performance on the instances in Set 1.

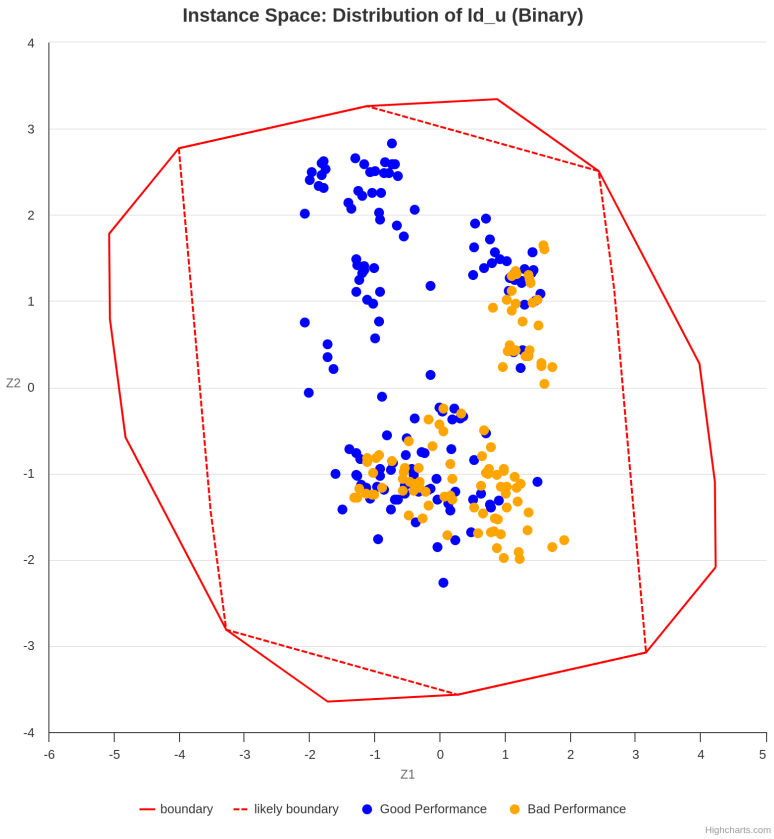

Figure 17: Binary distribution of uninitialized I_D performance on the instances in Set 1.

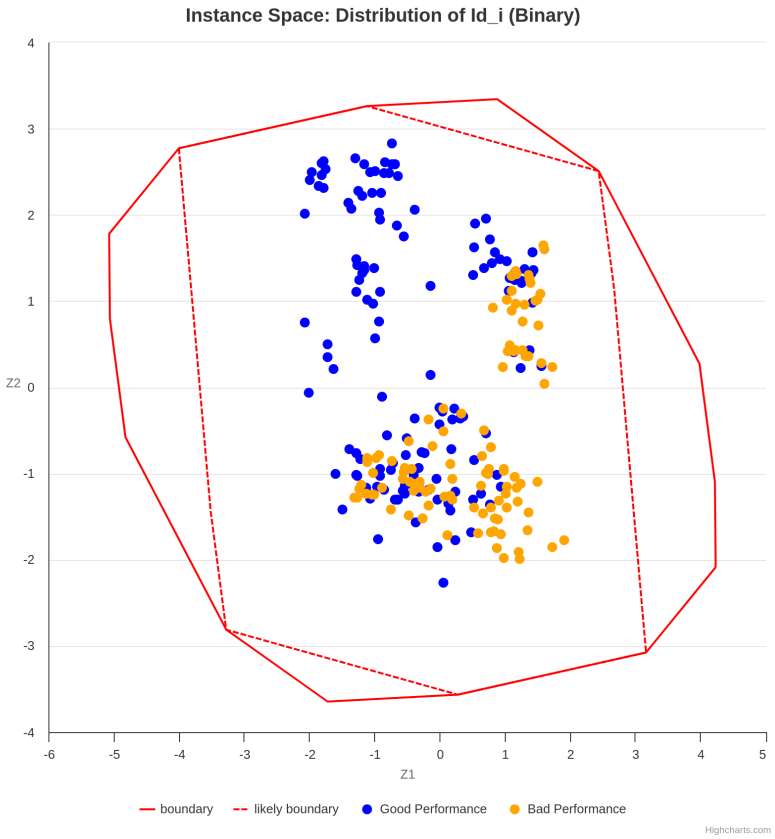

Figure 18: Binary distribution of initialized I_D performance on the instances in Set 1.

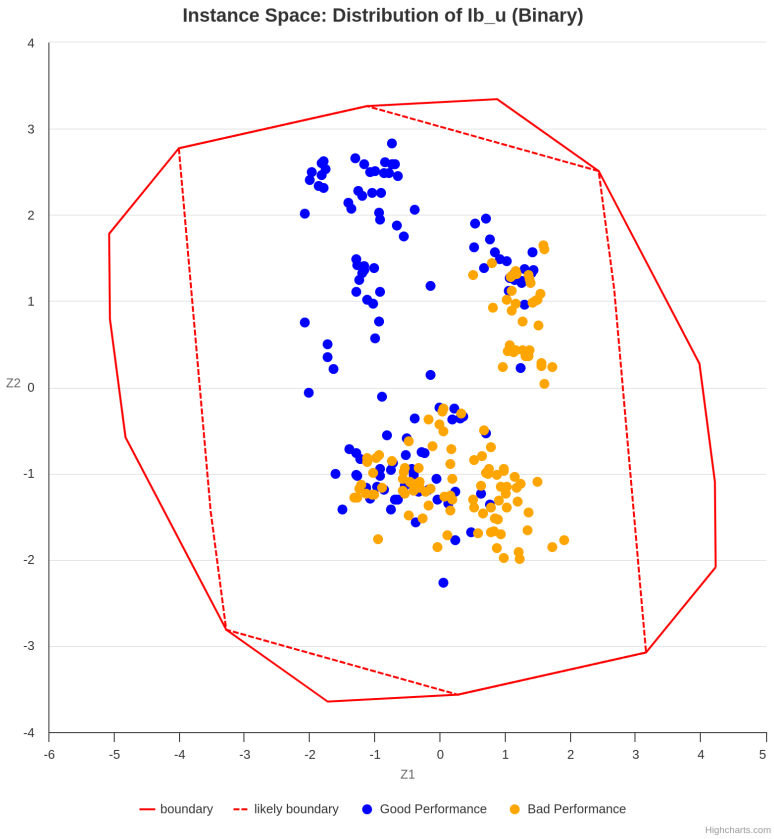

Figure 19: Binary distribution of uninitialized I_B performance on the instances in Set 1.

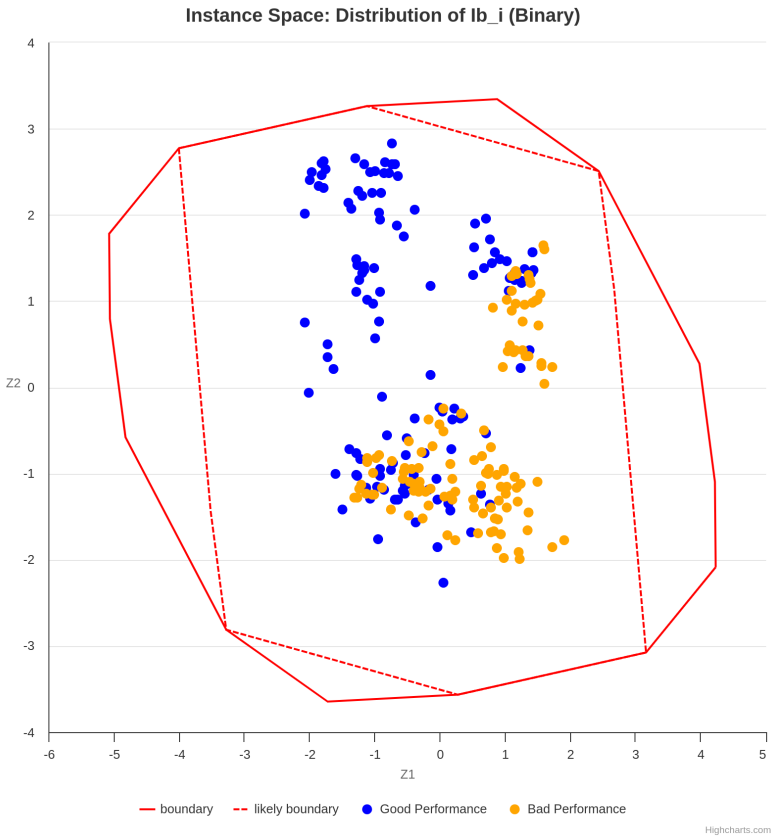

Figure 20: Binary distribution of initialized I_B performance on the instances in Set 1.

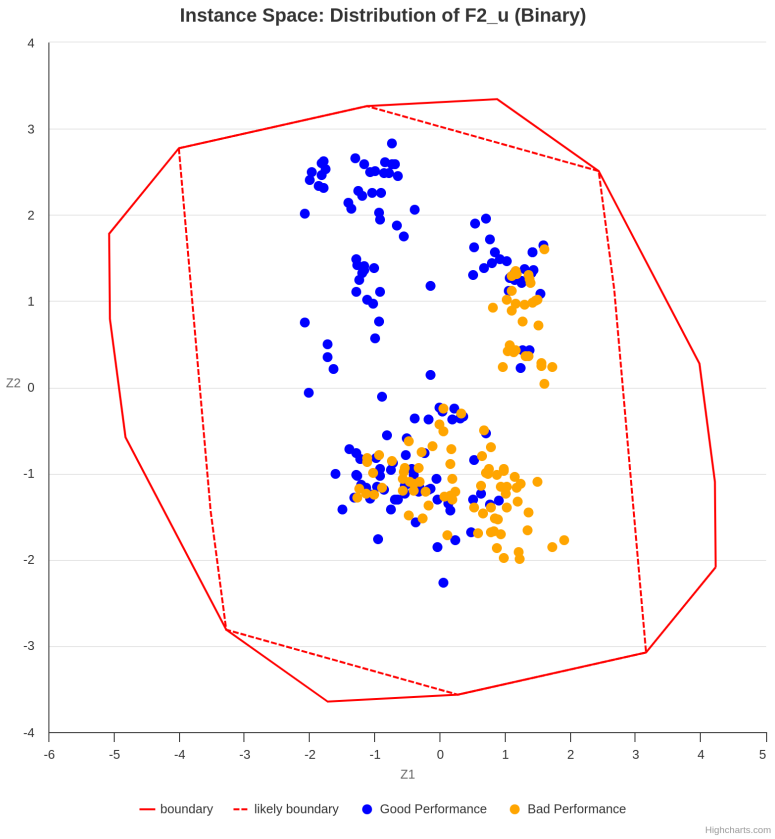

Figure 21: Binary distribution of uninitialized F2 performance on the instances in Set 1.

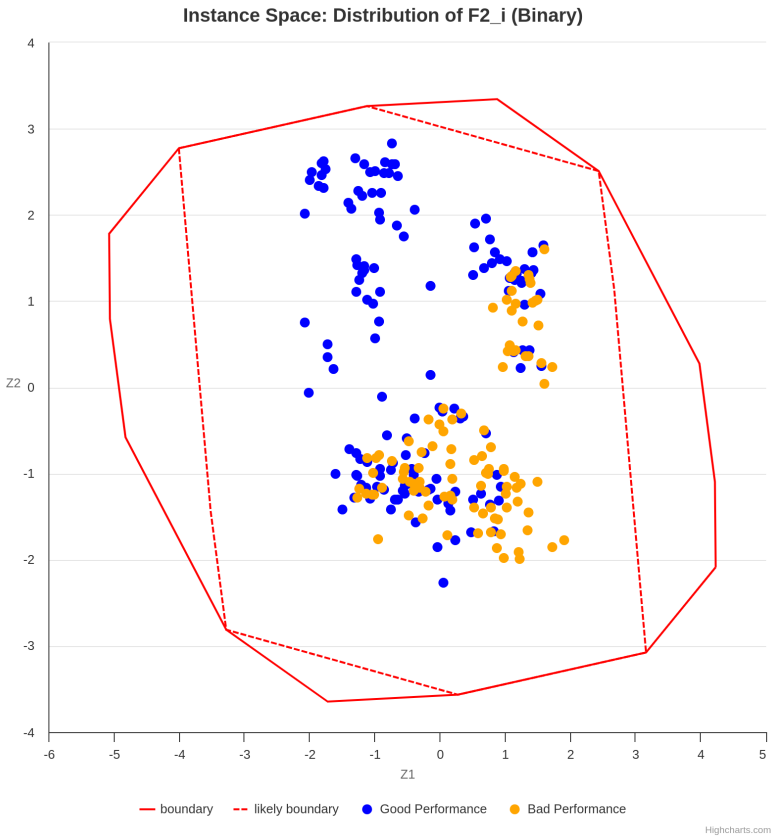

Figure 22: Binary distribution of initialized F2 performance on the instances in Set 1.

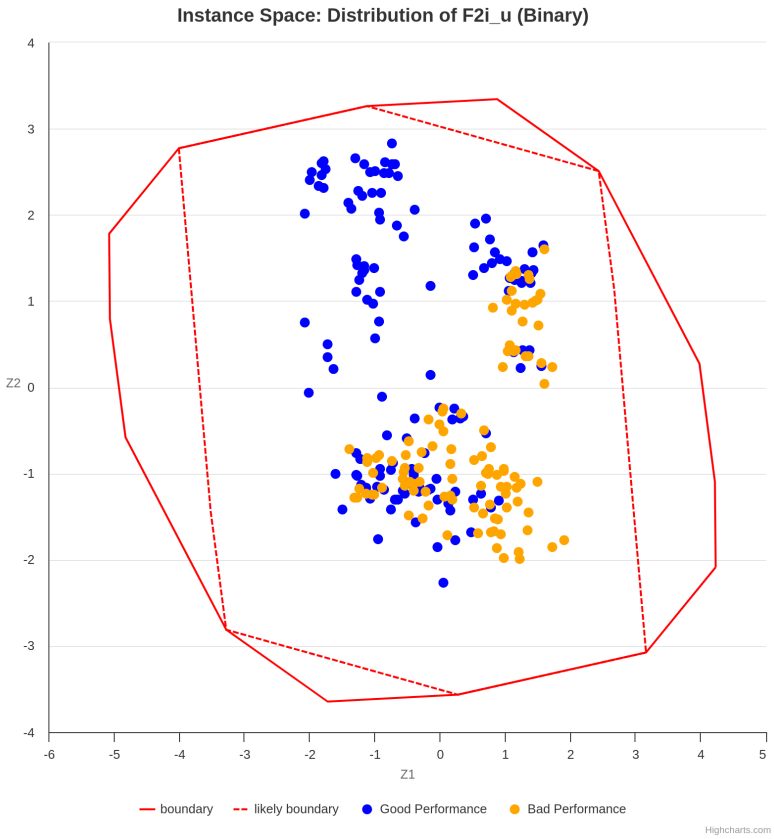

Figure 23: Binary distribution of uninitialized $F2_I$ performance on the instances in Set 1.

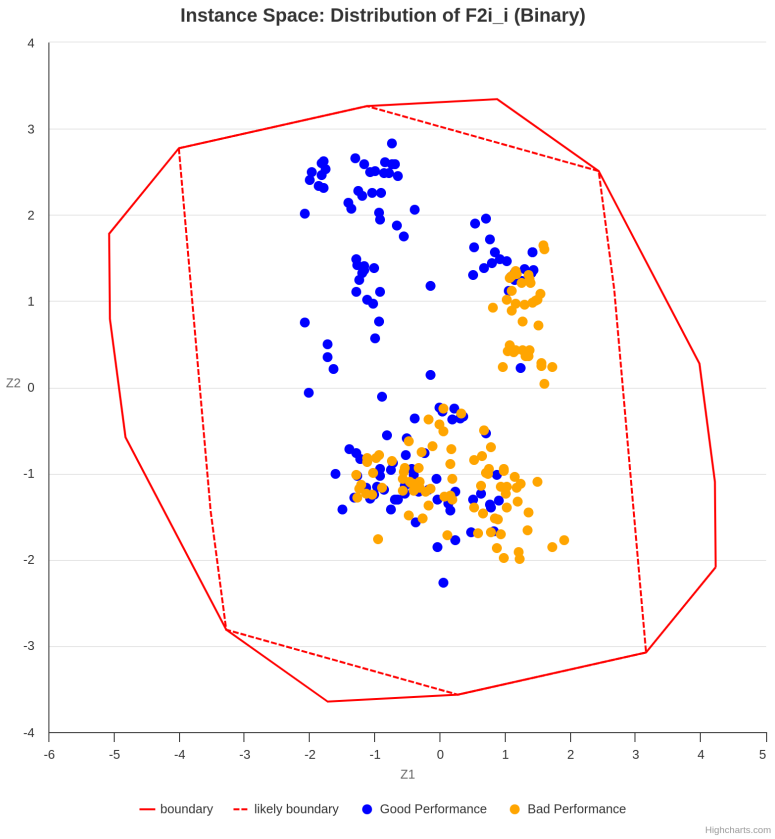

Figure 24: Binary distribution of initialized $F2_I$ performance on the instances in Set 1.

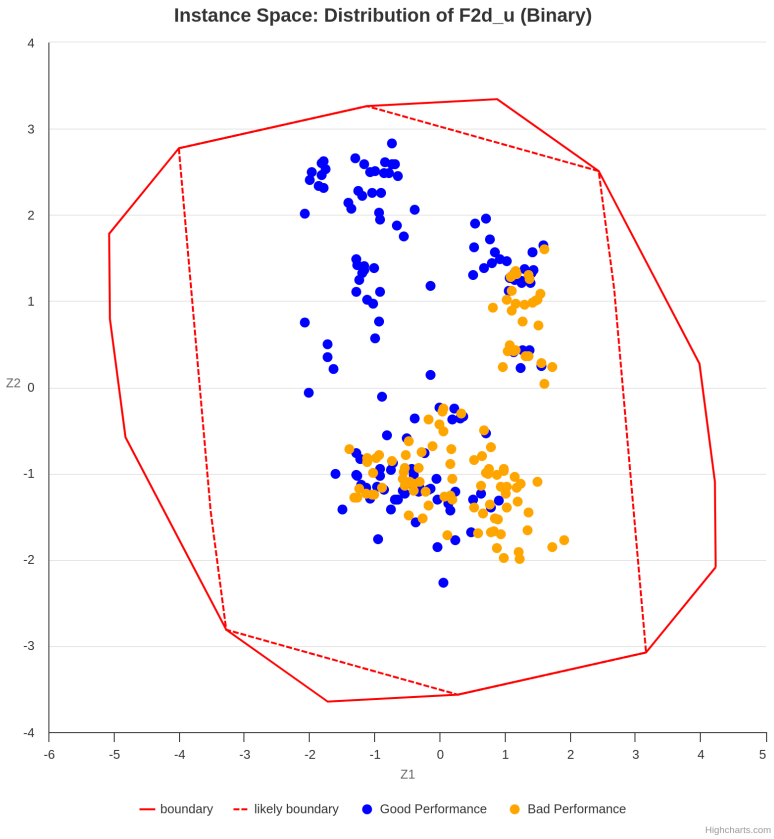

Figure 25: Binary distribution of uninitialized $F2_D$ performance on the instances in Set 1.

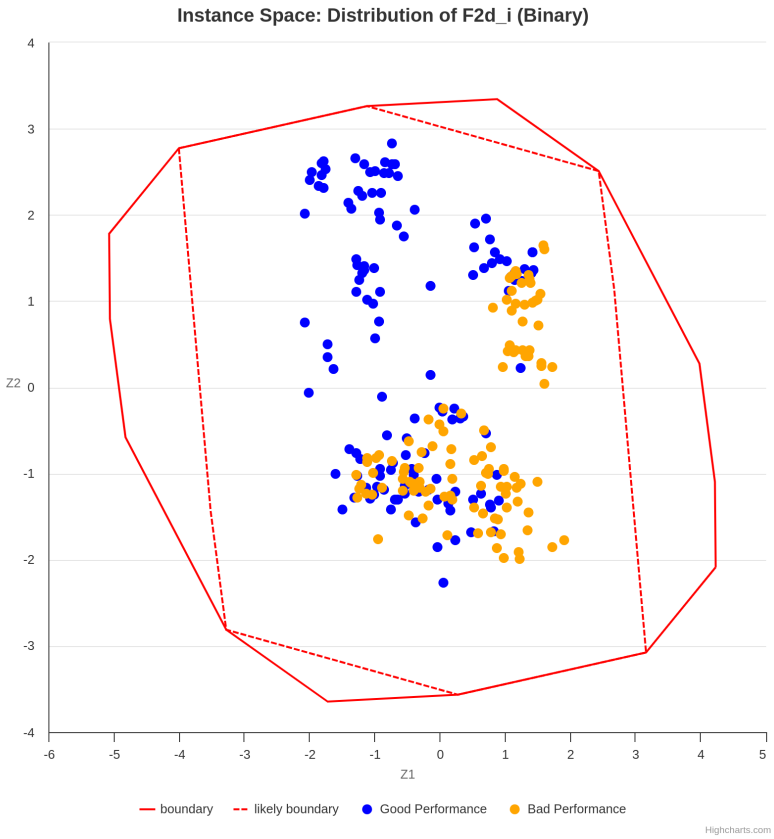

Figure 26: Binary distribution of initialized $F2_D$ performance on the instances in Set 1.

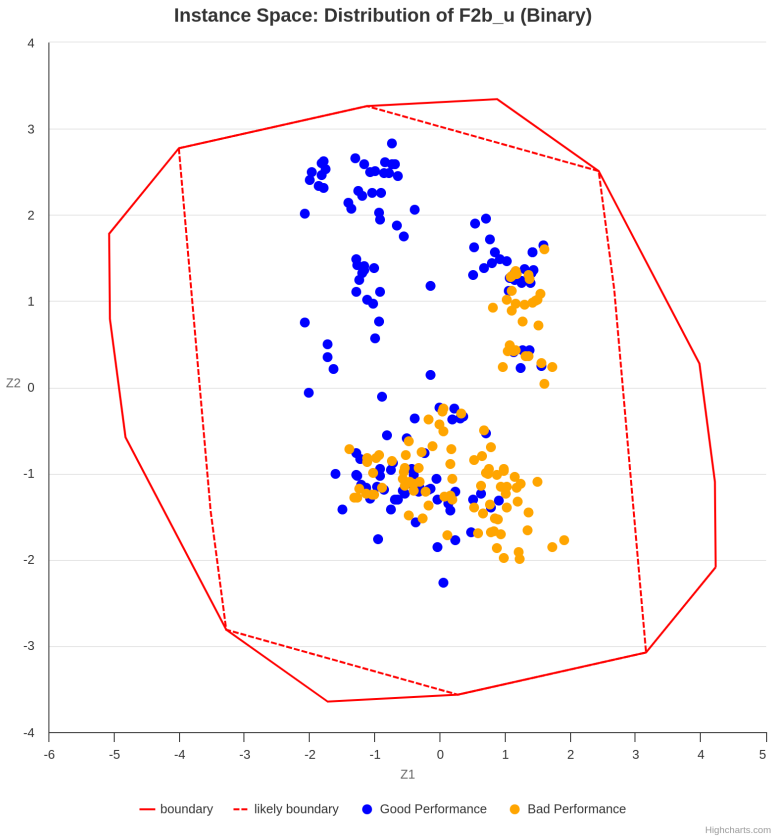

Figure 27: Binary distribution of uninitialized $F2_B$ performance on the instances in Set 1.

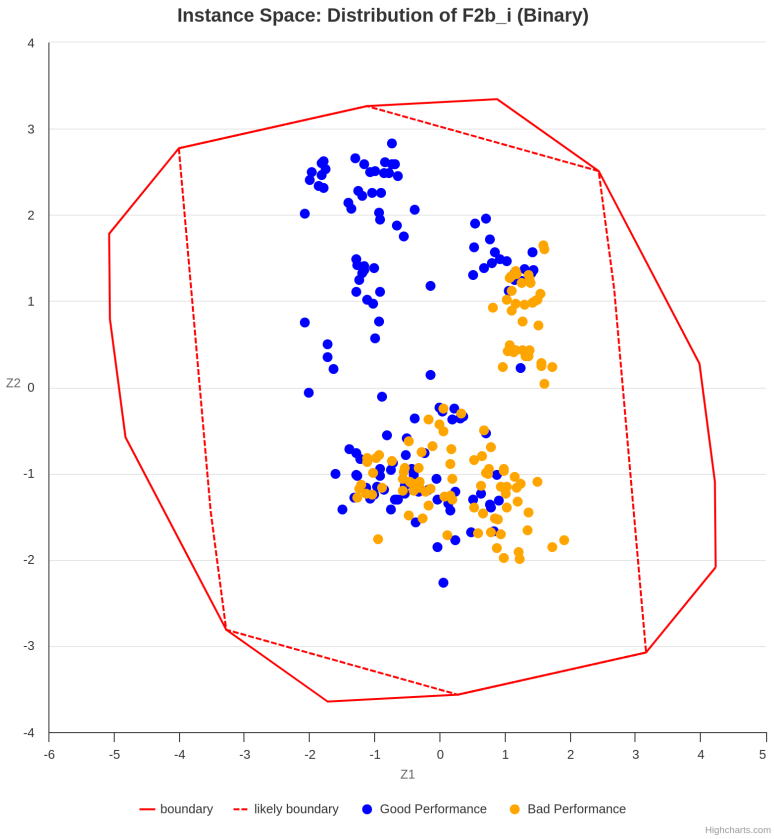

Figure 28: Binary distribution of initialized $F2_B$ performance on the instances in Set 1.

the conclusions outlined in the main report. By mapping features in this instance space, we gained insights into their relative importance and their impact on model performance.

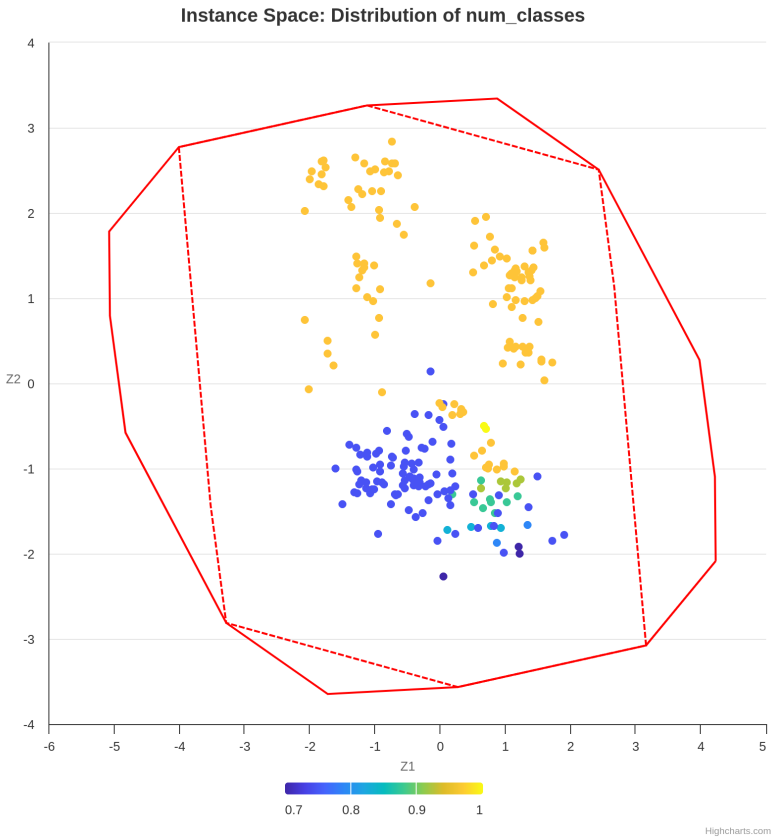

Figure 29: Distribution of the number of classes in the instances in Set 1.

Finally, Figure 36 confirms the results presented in Table 3 of the main report, highlighting formulation F1 with the heuristic initialization as the most effective exact approach. It is worth noting that no specific algorithm exhibits a clear advantage in some of the most challenging instances, leaving space for future research.

These visual representations not only serve as a valuable complement to the quantitative analysis presented in the main report but also provide a more intuitive understanding of the complex relationships between algorithm performance and feature distributions within the instance space. They offer a visual narrative that further supports the findings and conclusions drawn from the data.

Please refer to the following pages for detailed graphical representations of the performance and feature distributions.

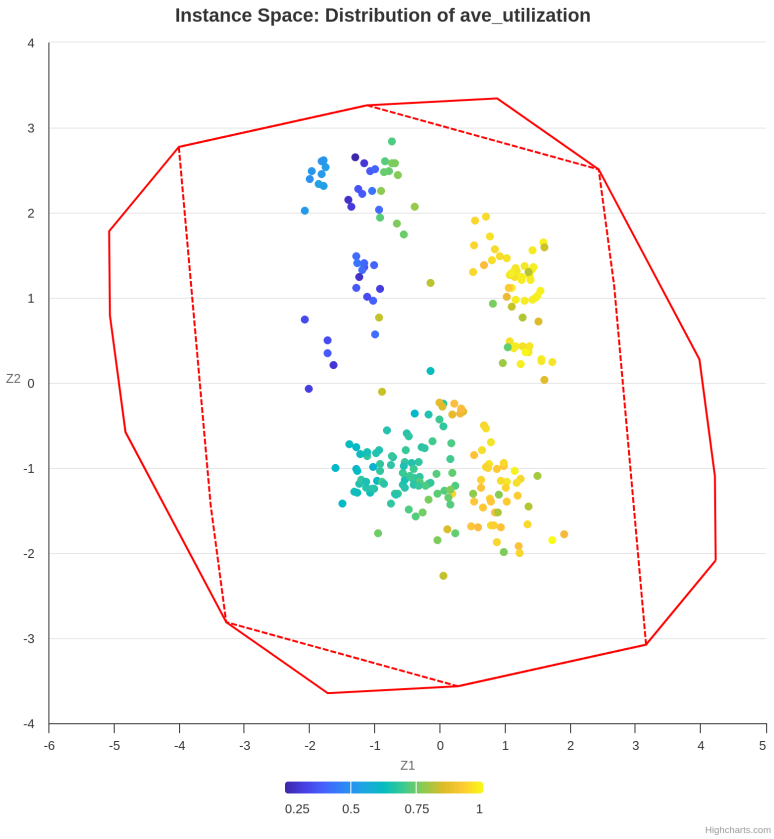

Figure 30: Distribution of the average of option utilization in the instances in Set 1.

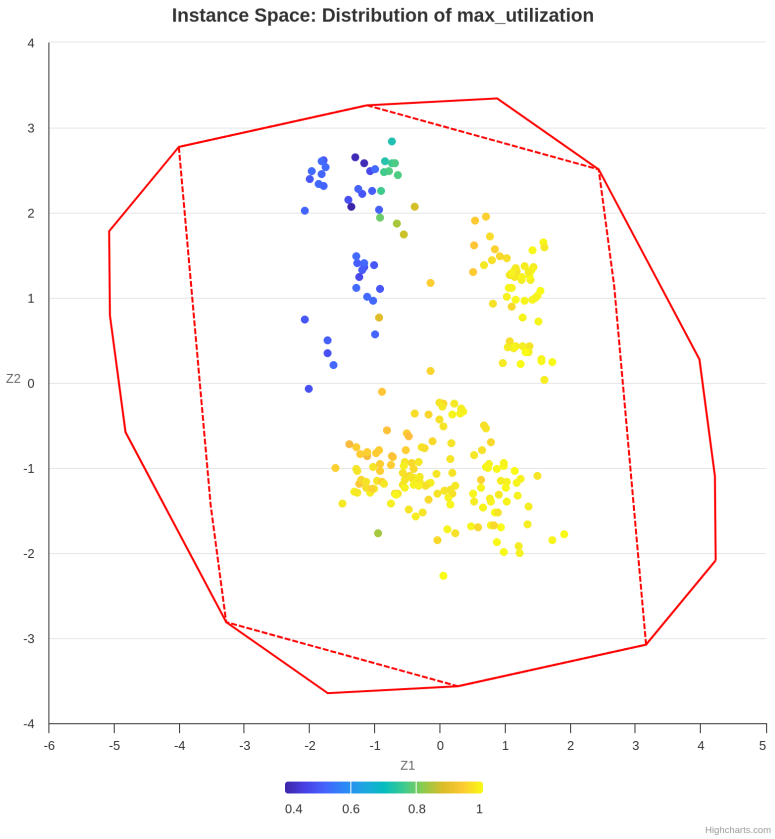

Figure 31: Distribution of the maximum of option utilization in the instances in Set 1.

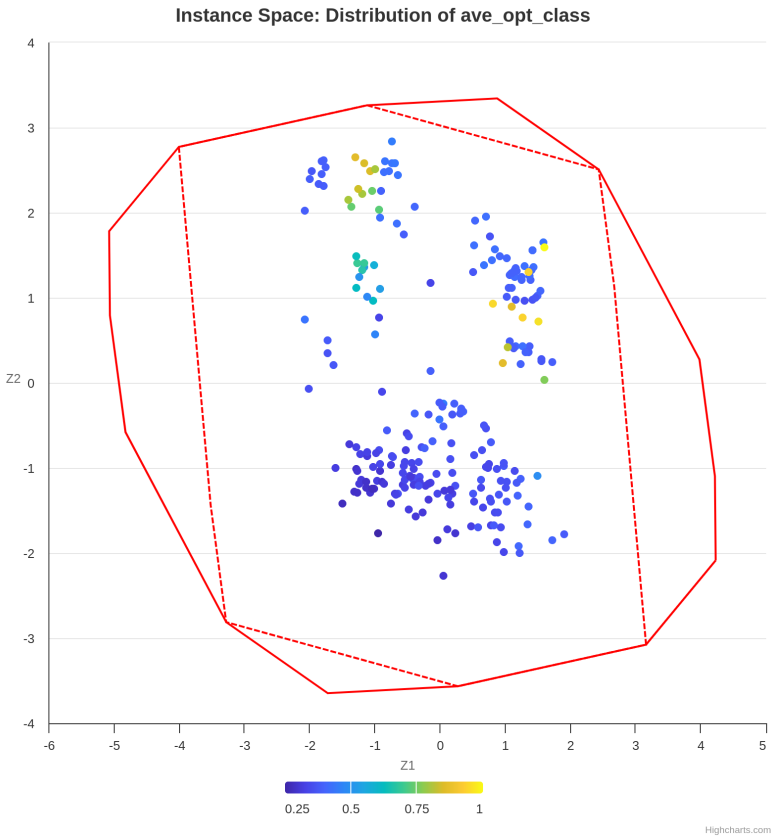

Figure 32: Distribution of the average number of options per car class in the instances in Set 1.

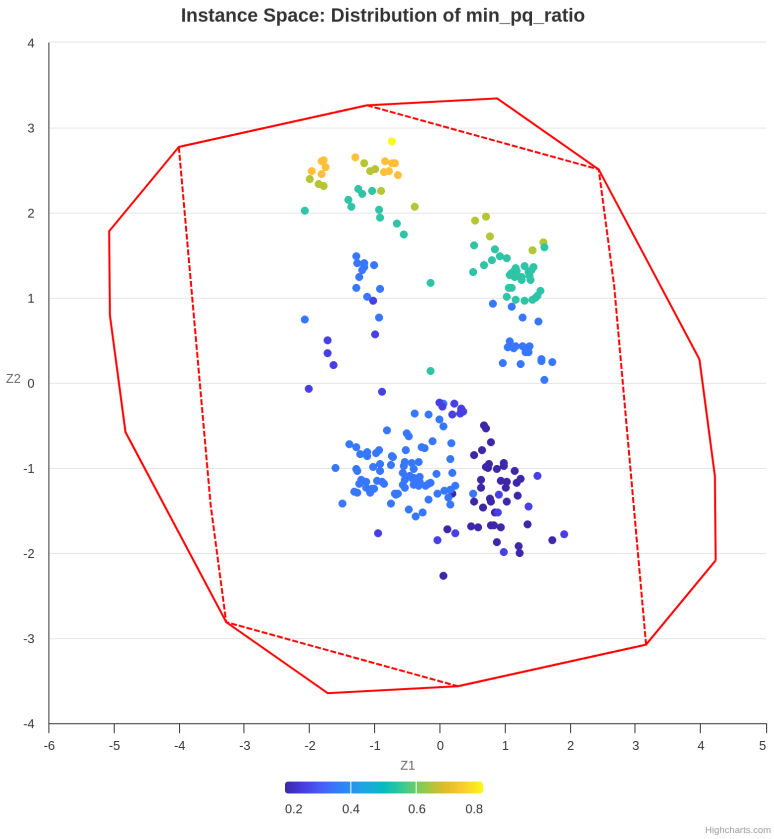

Figure 33: Distribution of the minimum of p/q ratio in the instances in Set 1.

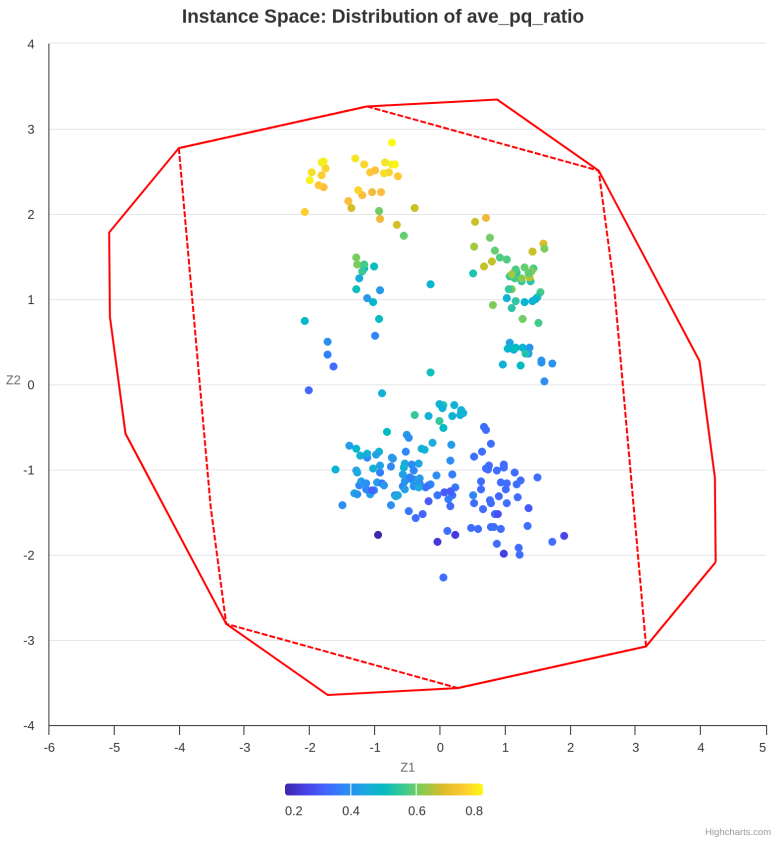

Figure 34: Distribution of the average of p/q ratio in the instances in Set 1.

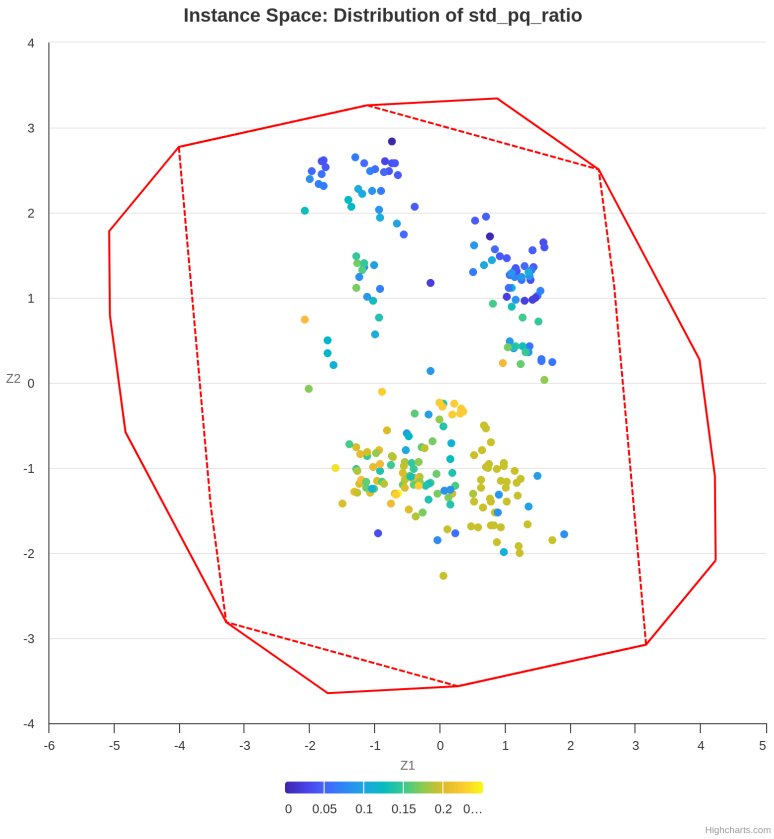

Figure 35: Distribution of the standard deviation of p/q ratio in the instances in Set 1.

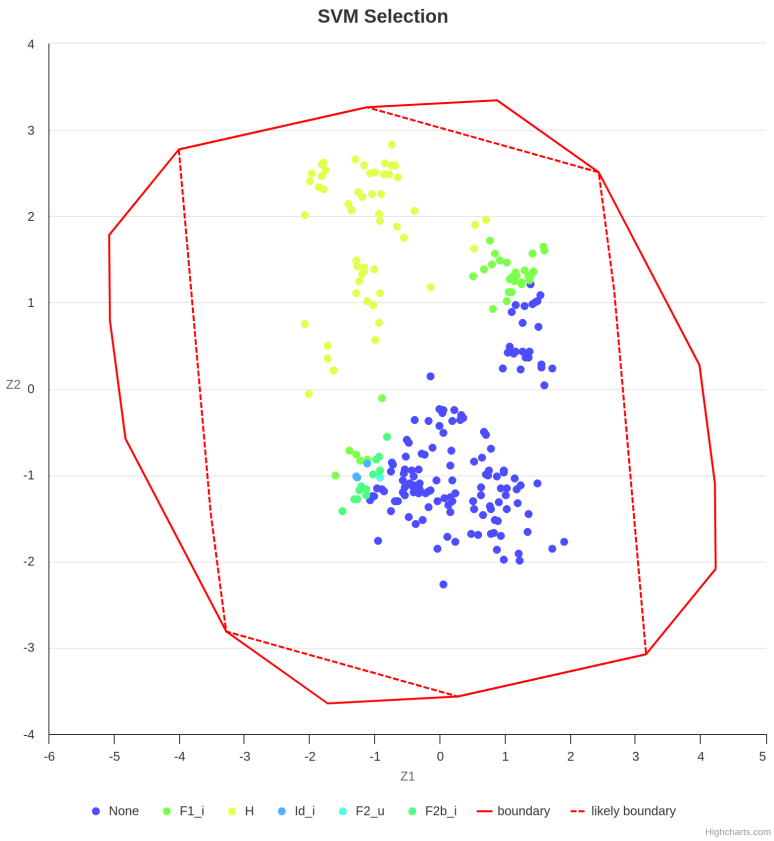

Figure 36: Best algorithm for each of the instances in Set 1.