Contributions to new variants of the open shop scheduling problem: modeling and solution methods

Levi Ribeiro de Abreu

UNIVERSITY OF SÃO PAULO
SÃO CARLOS SCHOOL OF ENGINEERING

Levi Ribeiro de Abreu

Contributions to new variants of the open shop scheduling problem: modeling and solution methods

São Carlos

2023

UNIVERSIDADE DE SÃO PAULO
ESCOLA DE ENGENHARIA DE SÃO CARLOS

Levi Ribeiro de Abreu

Contribuições para novas variantes do problema de programação da produção *open shop*: modelagem e métodos de solução

São Carlos

2023

Levi Ribeiro de Abreu

Contributions to new variants of the open shop scheduling problem: modeling and solution methods

Doctoral dissertation presented to the Graduate Program in Production Engineering from the São Carlos School of Engineering, University of São Paulo, to obtain the degree of Doctor in Science.

Concentration Area: Process and Operations Management

Advisor: Prof. Dr. Marcelo Seido Nagano

CORRECTED VERSION

São Carlos
2023

Catalog card prepared by Patron Service at “Prof. Dr. Sergio Rodrigues Fontes Library” at EESC/USP

```
        Abreu, Levi Ribeiro de
A162c      Contributions to new variants of the open shop
        scheduling problem: modeling and solution methods / Levi
        Ribeiro de Abreu; advisor Marcelo Seido Nagano. -- São
        Carlos, 2023.


           Doctoral (Dissertation) – Graduate Program in
        Production Engineering and Concentration area in Process
        and Operations Management. -- São Carlos School of
        Engineering, at University of São Paulo, 2023.


           1. Mathematical modeling. 2. Approximation algorithms.
        3. Population algorithms. 4. Open shop. I. Title.
```

Prepared by Eduardo Graziosi Silva- CRB-8/8907

FOLHA DE JULGAMENTO

Candidato: Engenheiro **LEVI RIBEIRO DE ABREU**.

Título da tese: "Contribuições para novas variantes do problema de programação da produção open shop: modelagem e métodos de solução".

Data da defesa: 22/03/2023

Comissão Julgadora	Resultado
Prof. Dr. **Marcelo Seido Magano** (Orientador) (Escola de Engenharia de São Carlos/EESC-USP)	Aprovado
Prof. Dr. **Anand Subramanian** (Universidade Federal da Paraíba/UFPB)	Aprovado
Prof. Dr. **Denis Borenstein** (Universidade Federal do Rio Grande do Sul/UFRGS)	Aprovado
Prof. Dr. **Marcus Rolf Peter Ritt** (Universidade Federal do Rio Grande do Sul/UFRGS)	Aprovado
Prof. Dr. **Matthias Thurer** (Chemnitz University of Technology/TU Chemnitz)	Aprovado

Coordenadora do Programa de Pós-Graduação em Engenharia de Produção:

Profa. Dra. **Janaina Mascarenhas Hornos da Costa**

Presidente da Comissão de Pós-Graduação:

Prof. Titular **Carlos De Marqui Junior**

Este trabalho é dedicado à minha mãe Meire Rose, cujo empenho em me educar sempre veio em primeiro lugar. Aqui estão os resultados dos seus esforços. Perdão pelos momentos de ausência. Com muito amor e gratidão, do seu filho, Levi.

AGRADECIMENTOS

A Deus, que me deu o privilégio e a oportunidade de chegar até aqui.

Agradeço à Universidade Federal do Ceará (UFC) pela minha graduação como Engenheiro de Produção Mecânica.

Ao meu orientador Nagano, que muito me apoiou durante minha trajetória no doutorado e que, apesar da distância e do período de pandemia, sempre prestou ajuda e orientação em todos os momentos em que estive escrevendo minha tese. Sua criatividade, sua disponibilidade e sua paciência foram essenciais para a conclusão desse trabalho, agradeço também por todas as conversas e orientações sobre os diversos aspectos da vida.

Agradeço aos professores do departamento de Engenharia de Produção da UFC, pelo empenho diário na transmissão dos conhecimentos do curso e, em especial, aos professores Albertin, Anselmo, Bruno, Heráclito, Moccellin (*in memoriam*) e Sergio (*in memoriam*) pelas orientações concedidas durante todo o período da graduação, pelo incentivo e pela motivação para eu seguir a carreia acadêmica.

Aos meus amigos do curso de Engenharia de Produção da UFC, Calixto, Ingrid, Joab, Jônatas e Roniel, e do Programa de Educação Tutorial (PET), meu muito obrigado por tudo. Certamente, com suas divertidas conversas e seus incentivos, me desenvolvi como ser humano, aluno e profissional de Engenharia de Produção.

Um agradecimento especial aos membros do laboratório de Otimização em Produção e Logística (OPL) da UFC, Kennedy e Vitor Hugo, pelo auxílio no desenvolvimento de modelos e algoritmos de fundamental importância para o progresso da tese, e aos professores Bruno, Framinan e Tavares por todas as orientações no desenvolvimento dos artigos provenientes desse trabalho.

Agradeço a minha mãe, Meire Rose, por moldar o meu caráter de maneira singular e por me apoiar com tamanho afinco ao longo dos meus anos de vida. Sem dúvida alguma, todos os postos que galguei até hoje foram alcançados pelas condições e pela dedicação incondicional propiciadas por ela.

Por fim, quero agradecer ao Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq - 831621/1999-0), à Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES - 001) e à Fundação de Amparo à Pesquisa do Estado de São Paulo (FAPESP - 2021/11586-2) por todo o apoio concedido para a realização desse trabalho.

"Que tudo seja feito com decência e ordem."
I Coríntios 14:40

ABSTRACT

ABREU, L. R. **Contributions to new variants of the open shop scheduling problem: modeling and solution methods**. 2023. 253p. Thesis (Dissertation) - São Carlos School of Engineering, University of São Paulo, São Carlos, 2023.

Several studies have been carried out regarding optimizing production scheduling in industrial environments. As a result, new variants related to several problems have been incorporated into this study area to cover the most diverse cases presented in productive environments. In this sense, this dissertation aimed to study a production scheduling problem little stressed in the literature, the production scheduling problem in an open shop environment with many applications in the industrial and services areas. In this study, we observed essential constraints for the described environment, which became new variants for the problem: the sequence-dependent setup times; the study of blocking machines in the processing of operations; the study of reprocessing or repetition of operations in the production process and the delivery of products through the vehicle routing. The objective of these problems was to minimize the total duration of the schedule (makespan). We proposed heuristic methods for modeling and solving these problems, such as priority rules, constructive techniques, bio-inspired meta-heuristics, and mathematical programming methods such as integer linear programming and constraint programming models and matheuristics. For the computational tests, we ran the methods with robust data from classical literature instances adapted to the constraints of the problems under consideration and new instances proposed during the study. The results showed that the proposed exact and approximate methods provided quality solutions with computational efficiency and were competitive compared to the literature methods.

Keywords: Mathematical modeling. Approximation algorithms. Population algorithms. Open shop.

RESUMO

ABREU, L. R. **Contribuições para novas variantes do problema de programação da produção *open shop*: modelagem e métodos de solução**. 2023. 253p.Tese (Doutorado) - Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos, 2023.

Nos últimos anos, diferentes estudos têm sido realizados no que tange à otimização da programação de tarefas em ambientes produtivos. Novas variantes, relacionadas a diferentes problemas, têm sido incorporadas a essa área de estudo a fim de se adequar ao maior número possível de ambientes reais de produção. Dentro desse contexto, a tese visou o estudo do problema de programação da produção em ambiente *open shop*. O *open shop* é um ambiente de produção pouco destacado na literatura, quando comparado aos ambientes clássicos de produção como o *flow shop* e o *job shop*, e possui diversas aplicações nos setores industriais e de serviços. Nesse estudo, foram observadas importantes características para o ambiente retratado, as quais se transformaram nas seguintes novas variantes para o problema: a consideração de tempos de preparação (*setups*) explícitos, dependente da sequência das operações e das máquinas; a existência de bloqueio de máquinas no processamento das operações; a existência de reprocessamento ou repetição de operações no processo produtivo; e a entrega dos produtos por meio da roteirização de veículos. O objetivo desses problemas foi encontrar soluções que minimizem algum indicador sobre o nível de serviço da operação, como a duração total da programação (*makespan*). Para a modelagem e a resolução desses problemas, foram utilizados métodos heurísticos: regras de prioridades; técnicas construtivas e meta-heurísticas bioinspiradas; e métodos de programação matemática, como modelos de programação linear inteira e de programação por restrições e mateurísticas. Para os testes computacionais, os métodos foram executados com dados robustos, advindos de clássicas instâncias da literatura adaptadas para as restrições dos problemas em consideração ou instâncias novas propostas durante o trabalho. Os resultados mostraram que os métodos exatos e aproximados forneceram soluções de alta qualidade e com eficiência computacional, sendo competitivos quando comparados com os atuais métodos da literatura.

Palavras-chave: Modelagem matemática. Algoritmos de aproximação. Algoritmos populacionais. *Open shop*.

LIST OF PUBLICATIONS

I. A new efficient biased random key genetic algorithm for open shop scheduling with routing by capacitated single vehicle and makespan minimization. **Engineering Applications of Artificial Intelligence**. 2021.

II. A new variable neighborhood search with constraint programming search strategy for open shop scheduling problem with repetitions of operation. **Engineering Optimization**. 2021.

III. New efficient heuristics for scheduling open shops with makespan minimization. **Computers & Operations Research**. 2022.

IV. A new hybridization of adaptive large neighborhood search with constraint programming for open shop scheduling with sequence-dependent setup times. **Computers & Industrial Engineering**. 2022.

V. A new two-stage constraint programming approach for open shop scheduling problem with machine blocking. **International Journal of Production Research**. 2023.

LIST OF FIGURES

LIST OF TABLES

CONTENTS

1 INTRODUCTION

The industrial environment is developing at a fast rate. Such changes are represented by the increasing levels of production provided by the constant technological advancement of methods and processes related to operations management. The production processes are becoming more specific and detailed, which allows them to operate on a large scale with personalized demands or demands with more flexibility (FRAMINAN; LEISTEN; GARCÍA, 2014).

In order to maintain the required production level in the industry, quantitative methods should be created and applied to support decision-making processes to achieve greater effectiveness of production systems, hence ensuring that the demands required of industry are delivered in the best possible way.

One of the study areas of industrial engineering where optimization methods for production systems are more applicable and with evident results is production scheduling. Production scheduling can be defined as determining where and when the necessary operations for a product will be allocated and carried out, defining dates to start and complete the necessary operations (PINEDO, 2016; FRAMINAN; LEISTEN; GARCÍA, 2014).

This area is dedicated to optimizing the allocation of tasks to available resources in a production system during a scheduling horizon. In this way, it is possible to make better decisions to improve classic problems in the industrial environment, whether large, medium, or small size, such as reduction of lead time, reduction of setup times, and better utilization of bottleneck resources (PINEDO, 2016).

With more detailed knowledge about the industrial environment, it becomes possible to realize the most different approaches to the problems already studied in production scheduling. For more complex production environments, such as the open shop, there is the opportunity for improvement in several approaches already presented in literature and even the development of new approaches for new variants of the problem in the face of the characteristics and needs present in the real world. Therefore, studying different production environments and their computational modeling is necessary, aiming to consider constraints that model the real systems to develop efficient resolution strategies. The present project proposes solution methods for the open shop scheduling problem. The problem is not so addressed in the literature when compared to other classical problems of the area (ANAND; PANNEERSELVAM, 2016).

This doctoral dissertation aims at proposing models and solution approaches for new emergent variants of the open shop scheduling problem. The problem is not so covered in the literature of scheduling when compared to other classical problems in the area (AHMADIAN *et al.*, 2021a). The objective is to minimize the scheduling duration (makespan), using for solving the problem: exact methods, such as mathematical and constraint programming models, and approximate methods, such as constructive heuristics, metaheuristics, and matheuristics.

The next sections 1.1 and 1.2 illustrate the motivation behind this research and the general and specific objectives. The remainder of this project dissertation is organized as follows: Chapter 2 explains the production scheduling problems and the notations for representing of these problems. Chapter 3 illustrates the main solution methods used for solving the open shop and their proposed variants, building up the theoretical foundations of the research. Chapter 4 shows the integrative summary of the new variants of the open shop described throughout the doctoral dissertation. Chapters 5, 6, 7, 8 and 9 illustrate the main contributions for each variant studied in the research, with each chapter representing a complete article about one variant. Finally, chapter 10 maps out the overall conclusions of each study and illustrates future research directions.

1.1 Motivation

Production scheduling problems are widely studied as optimization problems due to several industrial applications. The classic open shop production scheduling problem (OSSP) consists in scheduling a set of jobs on a set of machines, where each operation is associated with a processing time, not existing any predefined sequence of operations execution. The problem has practical and theoretical importance and has received less attention compared to the other classical production scheduling problems (ANAND; PANNEERSELVAM, 2016; ADAK; AKAN; BULKAN, 2020; AHMADIAN *et al.*, 2021a).

Since in the problem there is no predefined sequence of operations, the number of feasible solutions is more significant than classical production scheduling problems such as flow shop and job shop (AHMADIAN *et al.*, 2021a). The OSSP has many industrial applications, such as plastic injection, chemical processes, oil industries, food production, and pharmaceuticals. In the service sector, this problem can be modeled to schedule medical services, vehicle maintenance, museum visits, and telecommunications services (GONZALEZ; SAHNI, 1976; LIN; LEE; PAN, 2008; VINCENT; LIN; CHOU, 2010; NADERI; ZANDIEH, 2014; NADERI; NAJAFI; YAZDANI, 2012; CANKAYA; WARI; TOKGOZ, 2019; SHAREH *et al.*, 2021).

In classical production scheduling research in the open shop environment, more complex constraints on job flows/inventory are not considered or are considered part of the processing time in the case of setup times. However, this simplification may increase the scheduling duration when the obtained solution is implemented in the industrial environment due to not considering these fundamental properties (ALLAHVERDI, 2015;

ANAND; PANNEERSELVAM, 2016).

Hence, when these emergent constraints are considered a parameter of the problem, it is possible to develop more assertive schedules. Therefore, by explicitly taking setup times and machine blocking constraints into account, for example, a given production scheduling problem can be solved more realistically with these important characteristics that have many industrial applications (ALLAHVERDI *et al.*, 2008).

Despite the theoretical and practical importance of the open shop, contributions in the literature to new variants of the problem are limited. These limitations are highlighted in some recent literature reviews on production scheduling problems with setup costs (ALLAHVERDI; GUPTA; ALDOWAISAN, 1999; ALLAHVERDI *et al.*, 2008; ALLAHVERDI, 2015), blocking constraints (HALL; SRISKANDARAJAH, 1996; MIYATA; NAGANO, 2019), integrated scheduling and distribution problems (WANG; GRUNDER; MOUDNI, 2015) and on open shop problems (ANAND; PANNEERSELVAM, 2016; ADAK; AKAN; BULKAN, 2020; AHMADIAN *et al.*, 2021a). These are the most recent gaps in the open shop literature.

In summary, the main motivation for this study is that there are few studies in the open shop problem literature with different industrial applications, such as machine blocking and operations repetitions characteristics. In addition, the need for more efficient methods for solving the open shop in the current literature is also a research motivation, as shown in the most recent open shop literature reviews (ADAK; AKAN; BULKAN, 2020; AHMADIAN *et al.*, 2021a).

1.2 Objective

The doctoral dissertation aims to study the open shop production scheduling problem, considering its classical structure, together with emerging variants, based on characteristics of production scheduling in complex industrial environments, such as explicit and sequence-dependent setup times; repetition of operations; zero buffer or machine blocking constraints and job delivery by vehicle routing. In each variant, the objective is to minimize the makespan.

1.2.1 Specific objectives

1. Implement new constructive heuristics to solve the classical open shop problem, comparing it with seminal heuristics such as priority rules and constructive heuristics presented in the literature;

2. Use matheuristic techniques with the proposed constructive heuristics, mixed-integer linear programming, and constraint programming to solve the variant of the problem

with setup times, considering randomly generated instances from the literature. Comparing the algorithms with the main state-of-the-art solving methods;

3. Propose a formal definition for the open shop problem with machine blocking and develop mathematical models of integer and constrained programming and hybrid exact methods to solve the problem.

4. Study the open shop scheduling problem, considering the possibility of repetition of operations, demonstrating properties of the problem, and developing exact and approximate solving methods;

5. Study the open shop problem, integrating the scheduling and distribution of orders by capacitated single vehicle and using exact methods and bio-inspired metaheuristics for the problem's solution;

With the structure of the objectives mentioned above, the doctoral dissertation is formatted as a collection of articles. Each chapter is an article with a new variant of the open shop problem with the proposed mathematical modeling and solution methods.

Based on the achievement of the declared objectives, it is possible to improve the literature of the open shop problem by proposing efficient solution methods for the proposed variants considering constraints of real industrial environments, such as the machine blocking constraints, order distribution, and operation repetition. Furthermore, it is possible to develop efficient solutions methods to the problems already mentioned in the literature, comparing benchmarking methods, such as the classical open shop problem and the variant with setup times.

1.3 Research methodology

Regarding the research methodology used, according to Lakatos and Marconi (2010), scientific research can be classified concerning its nature, approach, objective and technical procedure. This doctoral project is characterized by the scientific method as applied nature research to solve real problems with a quantitative approach through computational techniques as the main approach.

Concerning the research objective, the project is classified as an exploratory and explanatory study because from a literature study on problem-solving methods, approximate and exact methods will be developed to solve the proposed problems.

As for the technical procedure, the project can be classified as bibliographic and experimental research since the methods tested for solving the problems will be developed based on a survey of the main articles in the literature about the open shop problem and its variants.

In summary, according to the methodology for quantitative modeling research proposed by Bertrand and Fransoo (2002), the present project is a normative axiomatic research. Thus, the study finds the solution of idealized problems with a robust industrial relationship, compares different solutions for problems already presented in the literature, and searches for effective solutions for emerging open shop variants.

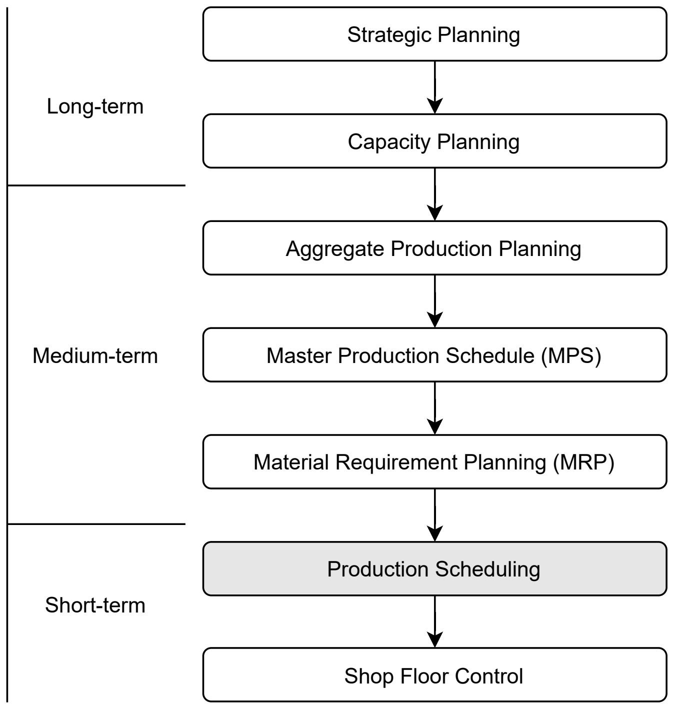
2 PRODUCTION SCHEDULING

Production planning, scheduling, and control (PPSC) is the area composed of several multidisciplinary activities that has the objective of commanding and coordinating the production process (TUBINO, 2017). The main activities performed by PPSC are shop floor control, demand forecasting, inventory and capacity management, aggregate planning, and **production scheduling**.

Production scheduling is a PPSC activity responsible for determining how tasks will be allocated to existing resources over time (MACCARTHY; LIU, 1993). This area consists of optimization problems with several industrial applications.

Figure 1 illustrates a diagram describing where is the production scheduling area in the PPCP phases. In addition, the figure illustrates the division of the planning phases for each of the PPSC stages into the short, medium, and long term.

Figure 1 – Diagram with phases of production planning, scheduling and control



Source: Adapted from Fernandes and Filho (2010)

As illustrated in Figure 1, long-term activities consist of the strategic and capacity planning of the operation. The medium-term consists of planning for job families, unit jobs, and material requirements (TUBINO, 2017). The production scheduling phase is in the short term of planning and is a more operational activity on the shop floor. The scheduling time horizon is usually weekly or daily. Therefore, the solution methods for the scheduling problems need low computational cost (PINEDO, 2016).

The classical production scheduling problems consist of a set of jobs $N \in \{1, 2, 3, ...,$ $n\}$, where n is the number of jobs, each job is processed on a set of machines $M \in \{1, 2, 3, ..., m\}$, where m is the number of machines, for each of the processing operations of the jobs $j \in N$ on the machines $i \in M$, there is a processing time p_{ij}. Furthermore, for each job $j \in N$, there is a release date r_j, the date on each job is available to be processed, and a due date d_j, the deadline for completion of all operations of jobs. These parameters are used to calculate the performance metrics of the proposed solutions (PINEDO, 2016).

The basic notation to define classical production scheduling problems was proposed by Graham *et al.* (1979). It consists of a tuple with three symbols $(\alpha|\beta|\gamma)$. The first symbol α indicates the machine environment of the problem, the second symbol β the technological constraints and job characteristics. Finally, the third symbol γ defines the performance measure(s) of the solutions to the problem.

The notation considers several different production scheduling environments; each environment describes how the machines are related to processing jobs. In addition, the notation describes the constraints involved, the objectives, and the job flows in the machines (MACCARTHY; LIU, 1993). The main α production environments are:

Single machine (1): an environment in which a single machine processes all jobs.

Parallel machines (Pm): there is only one production stage, with several machines operating in parallel. The machines can be identical (Pm), have different speeds (Qm), or unrelated processing times (Rm);

Flow shop (Fm): each job has the same processing sequence on the machines, corresponding to the production layout by-product or linear;

Job shop (Jm): each job has its sequence in the machines, resembling the production layout by-process or functional;

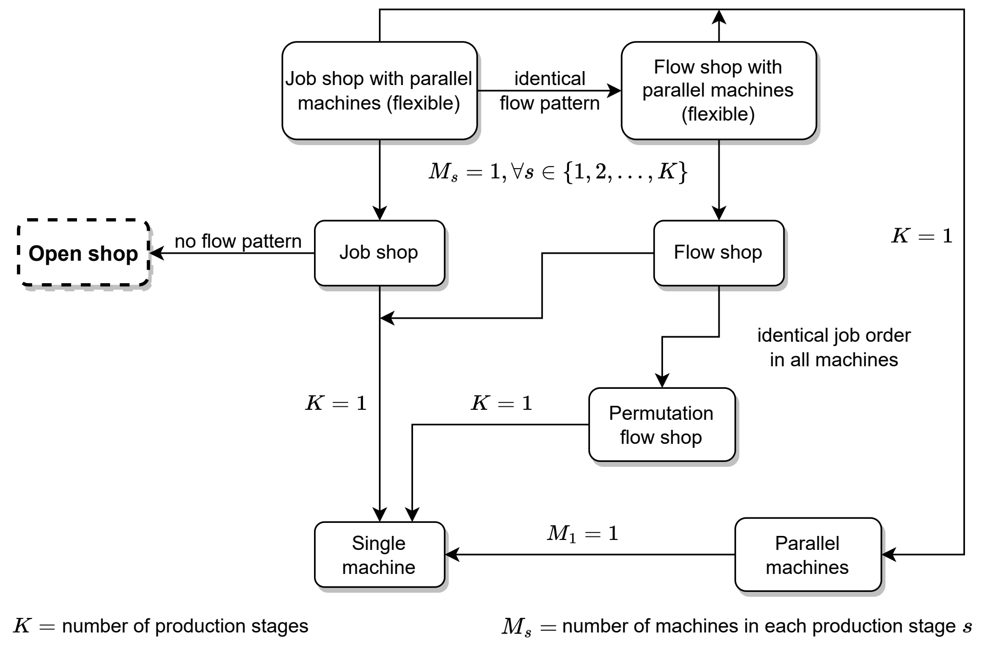
Open shop (Om): each job must be processed in all machines, and there is no specific sequence for each one of the operations. It is also called classic open shop;

Flexible flow shop (FFm): environment of flow shop machines where each production stage has a set of parallel machines;

Flexible job shop (JFm): job shop machine environment where each production stage has a set of parallel machines.

Figure 2 illustrates the relationship between the various classes of production scheduling problems, considering the differences in the production flow of jobs, the number of production stages in each environment, and the number of parallel machines present in each stage.

Figure 2 – Production environments and its characteristics


Job shop with parallel machines (flexible)
identical flow pattern
Flow shop with parallel machines (flexible)
$M_s = 1, \forall s \in \{1, 2, \ldots, K\}$
Open shop
no flow pattern
Job shop
Flow shop
$K = 1$
identical job order in all machines
$K = 1$
$K = 1$
Permutation flow shop
Single machine
$M_1 = 1$
Parallel machines
K = number of production stages
M_s = number of machines in each production stage s

Source: Adapted from Maccarthy and Liu (1993)

In addition to defining the environment of the production scheduling problem, it is necessary to illustrate the main parameters of these problems and what are the technological constraints of the industrial environment, which involve constraints on production flow, inventories, setup times, service level, etc. (PINEDO, 2016). Some of the main parameters and technological constraints β are:

Precedence ($prec$): each job may have a set of other jobs or operations that must be processed before the job is processed;

Blocking ($block$): the intermediate inventory between production stages cannot store jobs, so the jobs must wait for the next machine to be released to exit from the previous machine and perform their processing in the next machine;

Setup times (S_{sd}): between the processing of one job and another on a machine, there is a setup time to prepare the machine to process the new job. This time can be independent or dependent on the processing sequence.

Job family ($fmls$): jobs of the same family may have different processing times but may be processed on a machine, one after the other, without any setup time.

Machine eligibility(M_j): each job has a subset of machines where its operations can be processed.

Recirculation($rcrc$): occurs when a job can visit a machine or production stage more than once.

The performance measures are used to evaluate the quality of the generated solutions. These measures are the objectives of the problems. Some γ objective functions are illustrated below:

Makespan (C_{max}): total duration of the schedule or the finishing time of the last operation to complete the schedule;

Total completion time ($\sum C_j$): sum of the completion time of the processing of all jobs. The sum of the time that the jobs remain in the production system.

Maximum tardiness (T_{max}): the longest delay in the completion of each job concerning its due date.

Number of tardy jobs ($\sum U_j$): number of tardy jobs, where the completion time of at least one of its operations was longer than its due date.

Some of the production environments, technological constraints, and objective functions have been illustrated above. This chapter is a clipping of the main features of production scheduling problems. Other examples and variants of the parameters can be found in Maccarthy and Liu (1993) and Pinedo (2016).

3 THEORETICAL FOUNDATIONS

The theoretical foundation chapter will describe the types of exact and approximate solution methods to be used in the doctoral project, and a summary of these methods' state-of-the-art applied to the classical open shop problem. In addition, it will be described which forms of application of these methods have not yet been performed for open shop problems, constituting a gap to be addressed in research.

3.1 Mathematical programming

Mathematical and constraint programming are examples of exact techniques for solving optimization problems. These techniques aim to find the best solution to the problem, taking polynomial times for treatable problems. However, the solving effort can be exponential for more complex optimization problems, due to the difficulty of using these methods for large problems. The main characteristic of exact techniques is the existence of proof of optimality for the obtained solutions (ROTHLAUF, 2011).

Mathematical programming consists of modeling real-world problems through mathematical equations, seeking to maximize or minimize an objective function within a set of solutions that satisfy all constraints of the problem (HILLIER; LIEBERMAN, 2013). It is an example of an exact solution method.

A mathematical programming model consists of: decision variables, which constitute unknown values that will be calculated when solving the model; the parameters, which are previously known information about the problem; the objective function, which is the indicator that one wants to maximize or minimize; and a set of constraints, which delimit the space of solutions for the problem (ARENALES *et al.*, 2015).

Linear programming, a model in which all constraints and objective function must be linear, is a mathematical programming technique used to solve production scheduling problems by modeling linear equations with integer variables, using models with the decision variables can assume only discrete values (HILLIER; LIEBERMAN, 2013).

Mixed-integer linear programming (MILP), on the other hand, consists of a linear programming model with both continuous and discrete decision variables (ARENALES *et al.*, 2015).

The method to model production scheduling problems is generally through MILP models, due do simple modeling option of the decision variables with sequence or position-based notation (NADERI *et al.*, 2011b). A generic MILP model is illustrated below:

minimize

$$\sum_{i=1}^{n} c_i x_i + \sum_{j=1}^{p} d_j y_j \tag{3.1}$$

subject to

$$\sum_{i=1}^{n} a_{ki} x_i + \sum_{j=1}^{p} h_{kj} y_j \geq b, \qquad \forall k \in \{1, ..., m\} \tag{3.2}$$

$$x_i \geq 0, \qquad \forall i \in \{1, ..., n\} \tag{3.3}$$

$$y_j \in \mathcal{Z}^+, \qquad \forall j \in \{1, ..., p\} \tag{3.4}$$

The variables x and y are the continuous and discrete decision variables, respectively. The symbols c, p, and b are the problem's parameters. The symbols n and p are the quantities of decision variables, and, finally, the symbol m is the number of constraints of the problem. This problem can be solved using several algorithms such as decomposition methods, branch and bound, branch and cut, and dynamic programming (ARENALES *et al.*, 2015).

In summary, some of the literature has proposed solutions of the open shop through mathematical programming methods: Brucker *et al.* (1997) proposed a branch and bound algorithm for solving the classical problem, Guéret and Prins (1999) elaborated a new lower bound for solving the problem. In addition, the lower bound is important information of the problem domain. It can be used in heuristic and linear programming methods as a problem parameter to improve the exact methods.

As an example, Guéret, Jussien and Prins (2000) implemented an improvement of Bruckner's branch and bound using the information from ower bound. In addition, Ozolins (2019) proposed an exact solving method for the open shop problem using dynamic programming.

3.2 Constraint programming

Constraint programming (CP) is a modeling paradigm for solving combinatorial optimization problems, such as routing and scheduling problems (APT, 2003). It combines logic programming and constraint solving techniques to solve optimization problems. It is a relatively recent technique, when compared to linear programming, that has shown promising results in solving several problems, mainly in production scheduling problems, as in Trojet, H'Mida and Lopez (2011) and Lunardi *et al.* (2020).

A constraint programming model has the same linear mathematical programming model characteristics: parameters, objective function, decision variables, and constraints.

However, it is solved by exploring the problem domain, analyzing the constraints, and defining and setting values for the variables, with a tree search process (CHERRI, 2018).

Constraint programming in production scheduling problems has several advantages; due to the use of logic programming in modeling, it is possible to represent scheduling problems with a smaller number of constraints, using logical arguments, linear and non-linear operators. Therefore, it is possible to solve the models with reduced computational times.

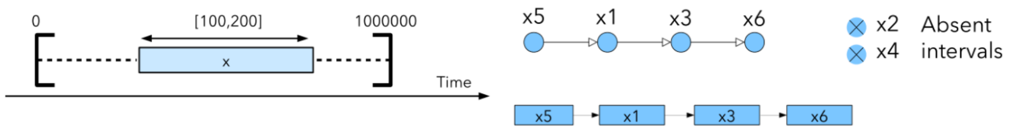
The main differences with linear programming are that constraint programming uses logical constraints for the problem and applies heuristic techniques to reduce the problem's search space. Moreover, constraint programming finds feasible solutions quickly due to the problem domain's exploration.

Another difference is that linear programming can represent problems with both continuous and discrete variables, making it easier to model mixed problems. On the other hand, constraint programming is capable of representing problems with only discrete variables, and it is difficult to model problems of a continuous nature (CHERRI, 2018).

Decision variables of constraint programming models, in some solvers, are of two types: interval and sequence variables. Interval variables represent an operation to be processed from a job on a machine, with the operation's start, end, and duration times. On the other hand, sequence decision variables group several interval variables into a set, such as the processing sequence of jobs on a machine. (LABORIE, 2018).

Figure 3 illustrates a schematic view of the two types of CP model decision variables. The interval variable (left) has a processing time of 100, starting at 100 and ending at 200. Its schedule horizon is between time 0 and 100000. The sequence variable (right) indicates the sequence of interval variables 5, 1, 3, and 6 processed in some problem resource.

Figure 3 – Interval (left) and sequence (right) decision variables of CP models

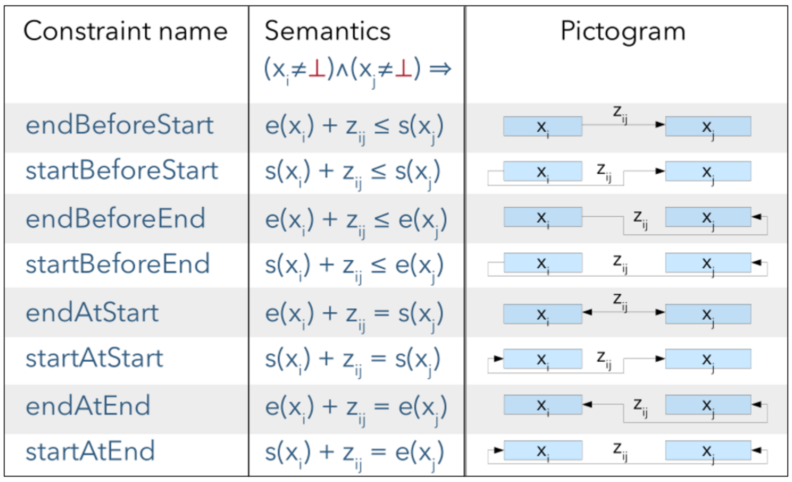


Source: Adapted from Laborie (2018)

The main constraints in CP models are the precedence constraints between two interval variables and the NoOverlap constraints of interval variables present in a sequence decision variables (LABORIE, 2018).

Figure 4 illustrates examples of precedence constraints used in CP models, where $e(.)$ is the end time of an x interval variable, $s(.)$ is the start time of an x interval variable, and z_{ij} is an optional wait time for processing an x_j variable after the x_i variable. In summary, the precedence constraints force interval variables to follow the established sequence with a specific waiting time as an optional parameter.

Figure 4 – Precedence constraints of CP models

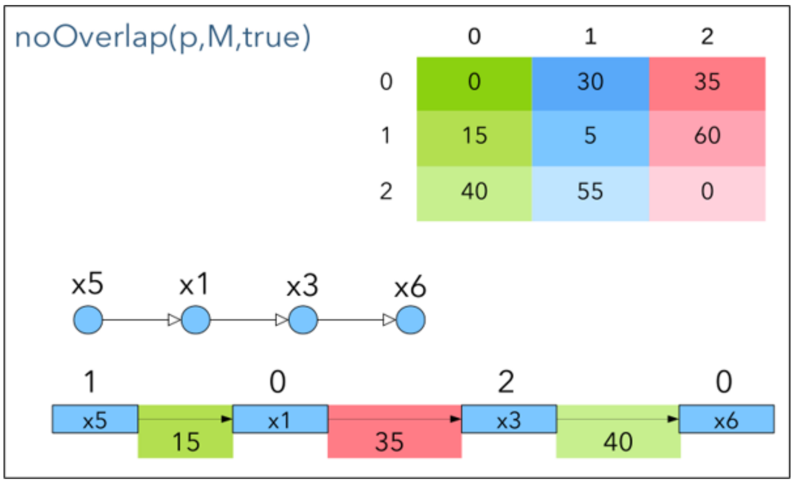


Constraint name	Semantics $(x_i \neq \perp) \wedge (x_j \neq \perp) \Rightarrow$	Pictogram
endBeforeStart	$e(x_i) + z_{ij} \leq s(x_j)$	x_i z_{ij} x_j
startBeforeStart	$s(x_i) + z_{ij} \leq s(x_j)$	x_i z_{ij} x_j
endBeforeEnd	$e(x_i) + z_{ij} \leq e(x_j)$	x_i z_{ij} x_j
startBeforeEnd	$s(x_i) + z_{ij} \leq e(x_j)$	x_i z_{ij} x_j
endAtStart	$e(x_i) + z_{ij} = s(x_j)$	x_i z_{ij} x_j
startAtStart	$s(x_i) + z_{ij} = s(x_j)$	x_i z_{ij} x_j
endAtEnd	$e(x_i) + z_{ij} = e(x_j)$	x_i z_{ij} x_j
startAtEnd	$s(x_i) + z_{ij} = e(x_j)$	x_i z_{ij} x_j

Source: Adapted from Laborie (2018)

Figure 5 illustrates a schematic view of how the NoOverlap constraint works in CP models, where p is the sequence variable, M is a transition matrix with the waiting time between two interval variables processed in sequence variable p, and *true* is an optional parameter that forces the processing of the next interval variable in p to be immediately after the end of the transition matrix time. NoOverlap constraints enforce that interval decision variables in a sequence decision variable are not executed at the same time, with a wait time if necessary (LABORIE *et al.*, 2018).

Figure 5 – NonOverlap constraints of CP models



	0	1	2
0	0	30	35
1	15	5	60
2	40	55	0

Source: Adapted from Laborie (2018)

Among the main competitive solvers for constraint scheduling, this doctoral project uses IBM's CP Optimizer, achieving competitive results for production scheduling problems and also packing and routing problems. (LABORIE; GODARD, 2007; LUNARDI *et al.*, 2020).

Table 1 shows an example of the main expressions used for modeling problems with the CP Optimizer solver. These expressions and constraints will be used in modeling the new variants proposed in the doctoral dissertation.

Table 1 – Expressions and global constraints used in the CP optimizer solver

Global constraints and expressions	
NoOverlap	it restricts a set of interval variables from overlapping each other in the same resource.
endAtStart	it limits to zero the delay between the end of one interval variable and the start of another.
Alternative	it forces an alternative constraint between interval variables.
endOf	it returns the end of one interval variable.
sizeOf	it returns the size of one interval variable.
presenceOf	it returns the presence status of an interval variable.
typeOfPrev	it returns the type of previous interval variable in an sequence.
startOfNext	it returns the start of next interval variable in an sequence.

Source: Authors.

Using the constraints and expressions in Table 1 it is possible to model many production scheduling problems logically, with few constraints and decision variables. For example, a simple CP model for a single machine problem with total completion time minimization is described below.

$$\text{minimize}$$

$$\sum_{j=1}^{n} \text{endOf}\,(x_j) \tag{3.5}$$

$$\text{subject to}$$

$$\text{noOverlap}\,(\Gamma)\,, \tag{3.6}$$

$$\text{interval } x_j, \quad \text{size} = p_j, \qquad \forall j \in \{1, ..., n\} \tag{3.7}$$

$$\text{sequence } \Gamma, \quad \text{on } [x_j]_{j \in \{1,...,n\}}\,, \tag{3.8}$$

$$\tag{3.9}$$

Equation (3.5) shows the objective function of the model with the sum of the completion time (endOf) of the operations of all the jobs in the single machine, represented by the interval variable x that each has the processing time with size p_j and n is the total number of jobs. Constraint (3.6) illustrates the non-overlap of operations x, through the sequence variable Γ, which has the sequence of jobs j on the machine. Thus the constraint forces that in the single machine, only one job is processed at a time. Finally, the constraints (3.7) and (3.8) illustrate the domain of sequence and interval variables.

Using constraint programming methods, we can cite: Malapert *et al.* (2012) proposed a constraint programming model for the classical Open Shop and Su and Hsiao (2015) used constraint programming to solve the problem considering eligibility and machine availability constraints, for the large instances, the method of Su and Hsiao (2015) had a large computational cost being outperformed by heuristic methods. Finally, Cankaya, Wari and Tokgoz (2019) implemented an integer and constrained linear programming model for solving the problem considering explicit setup times, with a case study application in chemical industry with tank cleaning.

3.3 Heuristic methods

Unlike exact methods, heuristic methods do not provide an optimal solution with proof of optimally for optimization problems, providing only approximate solutions to the problem. Nevertheless, heuristic methods have been used to solve production scheduling problems due to the speed of solution generation. This feature is essential for solving scheduling problems in complex industrial systems.

There are two main type of heuristics. Constructive heuristics start with an empty solution and, in each iteration, extend the solution until it becomes a complete and valid solution to the problem. On the other hand, improvement or local search heuristics start with a complete and valid solution and try to improve the solution quality through moves and perturbations in the neighborhood structure of the problem (PETCH; SALHI, 2003).

The priority rules are an example of a set of constructive heuristics to solve optimization problems in production scheduling. They are simple and easy-to-implement alternatives for production scheduling in real environments, always based on a performance measure, such as processing time, setup times, and job delivery dates (FUCHIGAMI; MOCCELLIN; RUIZ, 2015).

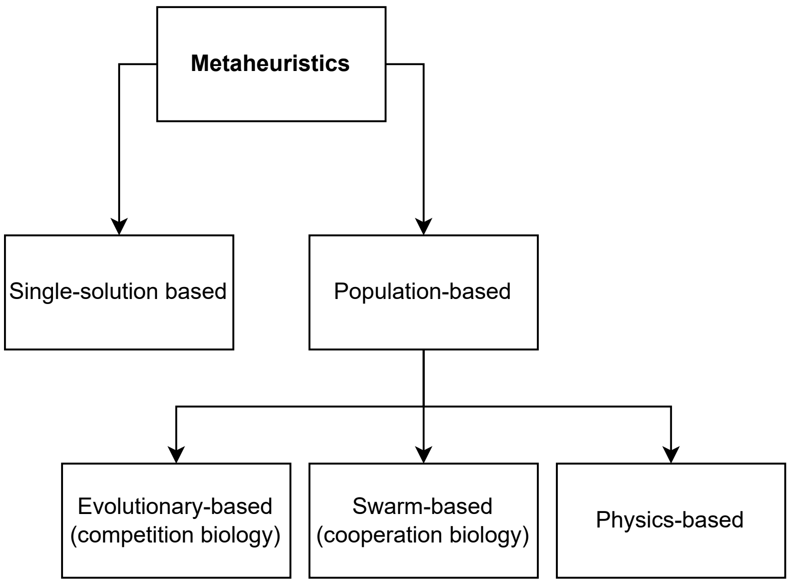
Regarding the application of heuristic methods for solving the open shop problem, there are the following application studies: Pinedo (2016) developed two constructive heuristics based on job processing times. Liaw (1985) proposed an improvement of Pinedo's heuristics with the insertion of solution improvement mechanisms. Naderi *et al.* (2010) implemented efficient heuristics based on redundant solution filters. Finally, Colak and Agarwal (2005) and Bai and Tang (2011) solved the classical problem using constructive greedy heuristics.

3.4 Metaheuristic methods

Other approximate methods for solving production scheduling problems are metaheuristics. In some cases, metaheuristics are inspired by nature, combining search tools with randomness to find optimal or near-optimal solutions.

Metaheuristics are general strategies for solving optimization problems, with features for intensifying the search and escaping local optima. Unlike heuristics that are problem-dependent, metaheuristics constitute general frameworks, and little effort is required to adapt their structure to a specific optimization problem (BLUM; ROLI, 2003). Figure 6 illustrates one type of classification of metaheuristics based on the number of solutions searched in each iteration of the algorithms.

Figure 6 – Classification of metaheuristics based on number of solutions


Metaheuristics
Single-solution based
Population-based
Evolutionary-based (competition biology)
Swarm-based (cooperation biology)
Physics-based

Source: Adapted from Salih and Alsewari (2020)

One of the main classifications of metaheuristics is the single-solution based, which search one solution at a time in each iteration, such as iterated local search or variable neighborhood search (VNS), Simulated Annealing (SA) and Tabu Search (TS). As the iterations occur, new solutions emerge, and the best solution is returned at the end of the algorithms. These methods, as showed in Figure 6, are also called single-solution metaheuristics.

It is possible to highlight single-solution metaheuristics to solve the open shop problem. For example, Liaw (1999) implemented a tabu search, Goldansaz, Jolai and Anaraki (2013) implemented an SA with an imperialistic strategy, Harmanani and Ghosn (2016) implemented an SA with a mechanism for efficiently exploiting the search space of the problem.

Population-based metaheuristics, on the other hand, can be bio-inspired, as in the competition of species in evolutionary strategies (evolutionary-based), in cooperation as in particle approaches (swarm-based), and in physics as in the behavior of electromagnetic particles (physics-based) (BLUM; ROLI, 2003).

The bio-inspired metaheuristics can be population-based and look to nature for inspiration to solve complex computational problems. As showed in Figure 6, theses metaheuristics based mainly on concepts of collective intelligence or evolution to generate multiple solutions and efficiently explore the search space. Bio-inspired algorithms have shown superior performance over single-solution metaheuristics in various production scheduling problems, such as: Andresen *et al.* (2008) and Soares and Carvalho (2020) to solve the open shop and parallel machine scheduling problems, respectively.

Bio-inspired metaheuristics can be based on the biology of competition and co-operation among species. Regarding competition, the evolutionary computing methods constitute genetic algorithms and evolutionary strategies. In these algorithms, a population of solutions exchanges genetic material among themselves, generating better offspring over generations, always keeping the best ones (GASPAR-CUNHA; TAKAHASHI; ANTUNES, 2012).

Regarding cooperation methods, there are collective intelligence or swarm-based methods, such as Ant Colony Optimization (ACO) and Particle Swarm Optimization (PSO). In these algorithms, a set of solutions is improved with the collective problem knowledge, and characteristics of the best solutions are shared among all solutions in the population (GASPAR-CUNHA; TAKAHASHI; ANTUNES, 2012).

Regarding physics methods, there is the study of Naderi, Najafi and Yazdani (2012) with an electromagnetism-like metaheuristic to solve the open shop problem.

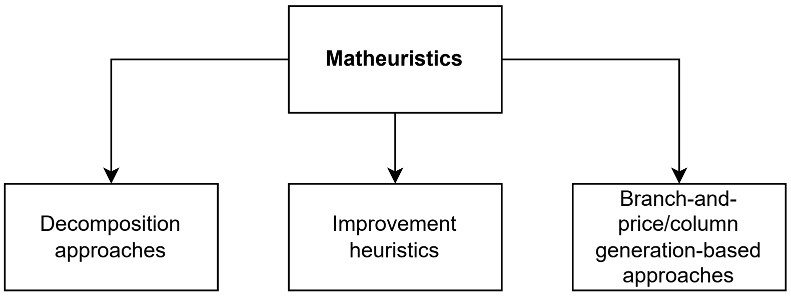
As an example of bio-inspired meta-heuristic methods, we can highlight the implementation of population algorithms to solve the open shop problem. Liaw (2000) implemented a hybrid genetic algorithm (GA), Blum (2005) implemented ant colony algorithm (ACO), Sha and Hsu (2006) implemented a particle swarm algorithm (PSO), Zobolas, Tarantilis and Ioannou (2009) developed a GA with a variable neighborhood structure, Anand and Panneerselvam (2018) implemented a GA with the steady-state strategy. Finally, Bouzidi, Riffi and Barkatou (2019) developed a cat swarm meta-heuristic for solving the problem.

3.5 Matheuristic methods

Matheuristics are hybrid strategies for solving optimization problems. The technique is defined as the hybridization of mathematical or constraint programming models into meta-heuristic algorithms or some heuristic approach (FISCHETTI; FISCHETTI, 2018).

Matheuristics can be used in problems where conventional models do not obtain reasonable solutions in admissible computational times. An essential feature of these methods is exploiting parts of the problem properties coming from mathematical or constraint models, with a meta-heuristic execution improving the exact method (MANIEZZO; STÜTZLE; VOSS, 2009). In summary, Figure 7 illustrates a classification of matheuristics proposed by Archetti and Speranza (2014) based on the problem handling approach.

Figure 7 – Classification of matheuristics based on type of problem handling



Source: Adapted from Archetti and Speranza (2014)

There are three main types of matheuristics: Decomposition approaches, improvement heuristics, and branch-and-price/column generation-based approaches. Decomposition approaches divide the problem into subproblems, and for each subproblem, either exact or heuristic methods are applied to solve it. Improvement heuristics hybridize metaheuristics with exact approaches. Exact approaches are used to improve solutions obtained by heuristic/metaheuristic methods as a local search procedure. Finally, Branch-and-price/column generation-based approaches change the main steps of these methods to decompose and solve integer models. The exact method is changed to speed up convergence, but the guarantee of optimally is lost. For example, the column generation phase is stopped prematurely to improve the solution process (ARCHETTI; SPERANZA, 2014).

Furthermore, these three approaches for matheuristics can be grouped or performed in parallel. For example, one can divide the problem into smaller subproblems and apply improvement heuristics with a local search through exact models in each subproblem. Hybrid matheuristics have obtained competitive performance in production scheduling problems (FISCHETTI; FISCHETTI, 2018).

As such, matheuristic is not a rigid paradigm (with defined terms and procedures) but a conceptual framework for designing mathematically useful heuristics. There are several application examples. Among the main ones is the proposal of interactive solving of relaxed and complete mathematical programming models with fixing subsets of decision variables or the use of metaheuristics during the execution of the solve. Both strategies aim at reducing the search space of decision variables (FISCHETTI; FISCHETTI, 2018).

There are no direct applications of matheuristics to open shop problems (AHMA-

DIAN *et al.*, 2021a). Recent applications of matheuristics methods in production scheduling and vehicle routing problems are: Ozer and Sarac (2019) developed MILP models and matheuristics for the parallel machine problem with shared resource constraints. Rohaninejad, Hanzálek and Tavakkoli-Moghaddam (2021) developed a hybridization of a genetic algorithm with a MILP as a local search for solving the problem of parallel machines with incompatible job families. Finally, Hà *et al.* (2020) developed a hybridization of constraint programming with single-solution metaheuristics for solving the vehicle routing problem with time windows.

4 INTEGRATIVE REVIEW

In the integrative review chapter, a summary of the state of the art for the open shop will be described, pointing out: the formal definition of the classical open shop, proposed new variants of the problem, and bibliometric analysis of the term open shop in the scientific databases. In addition, the role of each article (open shop variant) in the doctoral dissertation's construction will be explained, demonstrating at which stage of the evolution of the project dissertation each variant was inserted and what contribution each article made to the doctoral dissertation as a whole.

4.1 Classic open shop (OSSP)

Production scheduling problems in open shop environments are widely studied because of their many industrial applications. In recent decades, variants of the scheduling problems in flow shop and job shop environments have received much attention from researchers (see, for example, (FERNANDEZ-VIAGAS; RUIZ; FRAMINAN, 2017; FAN *et al.*, 2018; ZHANG; WANG; XING, 2019)) for recent reviews about permutation flow shop, hybrid flow shop, and job shop, respectively. However, this has not been the same for the open shop production scheduling problem(OSSP), which has received less attention from the researchers (ANAND; PANNEERSELVAM, 2016; ADAK; AKAN; BULKAN, 2020; AHMADIAN *et al.*, 2021a). This problem was first described by Gonzalez and Sahni (1976) and consisted of scheduling a set of jobs in a set of machines, where each machine-job operation has an associated processing time. However, unlike flow shop and job shop scheduling problems, in OSSP, there are no predefined routes for jobs on machines. In the notation by Lawler *et al.* (1993) , the problem is defined as: $O_m||C_{max}$.

When the objective is minimizing the makespan, and there is only one machine, the OSSP can be reduced to the single-machine scheduling problem, and any solution is optimal. For the two-machine case, polynomial algorithms with optimally proofs exist (GONZALEZ; SAHNI, 1976). However, for problems with three or more machines, the OSSP is NP-Complete (GAREY; JOHNSON, 2012).

Regarding the main exact methods for solving OSSP, three can be highlighted: the linear programming models with branch-and-bound as solution procedure, dynamic programming, and constraint programming (AHMADIAN *et al.*, 2021a). Although some branch and bound algorithms have been proposed for this problem (BRUCKER *et al.*, 1997; GUÉRET; PRINS, 1999; GUÉRET; JUSSIEN; PRINS, 2000), the exact MILP methods are rather limited to solve OSSP problem instances of realistic size.

Concerning constraint programming techniques, Malapert *et al.* (2012) proposed a new approach that incorporates recent constraint propagation techniques, integrating with knowledge of the problem's lower bound, restart, and search diversification approaches. The method is compared with metaheuristics and exact algorithms from the problem literature and got competitive results in classical benchmarking instances.

Finally, Ozolins (2019) developed an exact dynamic programming algorithm, being the first such approach applied to OSSP. Computational results show that the proposed algorithm can solve benchmark instances with moderate sizes.

Given the difficulty of solving OSSP with the makespan minimization objective, different approximate algorithms have been proposed. These are classified as constructive heuristics/local search or meta-heuristic approaches. Concerning constructive heuristics/local search, several contributions have been presented (PINEDO, 2016; GONZÁLEZ-RODRÍGUEZ *et al.*, 2010; COLAK; AGARWAL, 2005; NADERI *et al.*, 2010).

Regarding the application of metaheuristics for OSSP, the following stand out: Ahmadizar and Farahani (2012) propose a hybrid genetic algorithm with a local search optimization procedure that outperforms previously reported metaheuristics for OSSP. Ghosn, Drouby and Harmanani (2016) propose a parallel genetic algorithm using deterministic and random moves. Pongchairerks and Kachitvichyanukul (2016) propose a two-level PSO that works very well on literature instances. Finally, an extended genetic algorithm is proposed by Hosseinabadi *et al.* (2018) to solve the OSSP, being the most recent metaheuristic so far for the problem.

As can be seen from the above analysis, several methods provide approximate solutions to the OSSP with makespan minimization. However, there is still space for improving the state of the art of the problem by proposing new efficient constructive heuristics that incorporate some knowledge of the OSSP domain. More specifically, this doctoral dissertation suggests applying a mechanism that estimates the lower bound and makespan contribution of a partial solution in a constructive heuristic to insert operations to discard less promising solutions as a filter mechanism. Another proposed mechanism is an insertion algorithm that considers the idle time of machines between operations. Both mechanics are improvements with beam search and cheapest insertion search strategies.

These developed heuristics will be tested on classical instances in the literature and serve as a basis for developing metaheuristics and exact hybrid methods, both for the OSSP and variants of the problem proposed in this doctoral dissertation.

4.2 Open shop with explicit setup times (OSSPST)

Unlike other classical production scheduling problems, there is no previous sequence for the processing of the jobs in the machines in the open shop problem. Thus, the problem has a large number of feasible solutions. In the classical problem, each job can only be processed by a machine at a time, and the jobs, when their processing starts, cannot be interrupted (ABREU *et al.*, 2020).

The variant of the present dissertation considers that between the processing time of one job and one machine, there is a setup time for the operation that depends on the previous job processed on the machine. The setup time is also non-anticipatory because the job must already complete its processing on the previous machine to perform the setup. It is also necessary for the job to enter the next machine to perform the setup. The objective function is the makespan minimization. In Lawler *et al.* (1993) notation the problem has the form: $O_m|S_{sd}|C_{max}$

Allahverdi, Gupta and Aldowaisan (1999) developed a literature review (survey) of production scheduling problems with setup costs. For the open shop problem, only one paper had been proposed at that time. For the second survey performed, eight more articles were reviewed (ALLAHVERDI *et al.*, 2008). In the last one performed by the author, four more articles involving the problem were cited. Thus, based on the OSSPST articles quantity when compared to other scheduling problems, the topic has not yet been fully explored in the literature (ALLAHVERDI, 2015).

The first article to consider setup times in the open shop explicitly was Strusevich (1993), which considered the case for the open shop with two machines, setup times, processing times, and job removal times. In addition, the author developed an exact algorithm in polynomial time $O(n)$ for makespan minimization. Subsequent papers proposed exact strategies for solving special cases of the open shop with setup times, considering the problem with two machines or other trivial constraints (BEVERN; PYATKIN, 2016; BABOU; REBAINE; BOUDHAR, 2021).

The exact solving strategies define good properties for the knowledge of the problem. However, in some instances, these methods are optimal only for special cases of the open shop, simplifying the main characteristics such as small machine numbers, constant or machine-independent processing times, and fixed or independent setups. Therefore, for the problem proposed in this paper, arising mainly from real complex systems, it is necessary to use approximate strategies, such as heuristics and metaheuristics.

For solving problems with real characteristics, with large numbers of machines, jobs, and constraints, it is necessary to use approximate solving algorithms. These include priority rules, constructive solution strategies, and nature-inspired optimization algorithms.

Naderi *et al.* (2011a) developed an electromagnetic (EM) meta-heuristic for the open shop problem with sequence-dependent setup times, with minimization of the total completion time (TCT). The author tested the algorithm with a mixed-integer linear programming (MILP) model and obtained significant results, which were analyzed by statistical tests.

Abreu *et al.* (2020) devised a hybrid genetic algorithm for the open shop problem with sequence-dependent setup times and TCT minimization. In addition, three new constructive heuristics were also proposed, based on features of the problem domain, such as lower bound. Finally, the authors compared the GA with the EH of (NADERI *et al.*, 2011a) and the developed heuristics. Computational tests showed that GA obtained significantly better results than all other methods.

Mejía and Yuraszeck (2020) studied the open shop problem with sequence-dependent setup and transport times. A Variable Neighborhood Search (VNS) metaheuristic was proposed. The authors tested the algorithm with new and classical instances from literature, adapted for the problem, compared with another metaheuristic and four constraint programming models, with different search strategies. The methods were used to solve problems with makespan and TCT minimization. For both cases, VNS presented competitive results.

Despite existing contributions (notably the algorithms EH by Naderi *et al.* (2011a), GA by Abreu *et al.* (2020) and VNS by Mejía and Yuraszeck (2020)), there is space to develop more efficient solution procedures for the problem, based on the hybridization of exact and approximate methods and characteristics of the problem domain, such as the type of setup.

This project dissertation aims to present a hybridization of adaptive large neighborhood search with constraint programming (ALNS-CP) for OSSPST and makespan minimization as an objective. For an efficient generation of the initial solution, a constructive heuristic proposed by Abreu *et al.* (2020) is used, which produces reasonable solutions in competitive computation times. A new constraint programming (CP) model is used in the solution reconstruction phase as a solution reconstruction and local search strategy. The new model uses features of the problem domain, such as non-anticipatory setups, to reduce the number of variables and constraints.

4.3 Open shop with machine blocking (OSSPB)

In the literature, most of the production scheduling problems consider that the capacity of the intermediate buffer between machines is unlimited. That is when the process of a machine's current job finishes. The machine is always available to process the next job (HALL; SRISKANDARAJAH, 1996). However, in some practical environments, production processes deal with physical buffer limits, causing machines blocking from processing the next jobs (MIYATA; NAGANO, 2019).

According to Hall and Sriskandarajah (1996), one of the reasons for the occurrence of blocking is the lack of intermediate storage between machines or production steps. In addition, another possible cause of blocking lies in the production technology itself. For example, temperature or other characteristics of the materials require that the finished job remains on the machine to avoid deterioration or additional costs until the next machine is released. In addition, in industrial residues material handling, different types of waste are discharged from machines, preventing the work from leaving the machine until the process has finished (MIYATA; NAGANO, 2019).

The blocking assumption for the open shop, also called zero buffer constraints, means that intermediate storage of materials between adjacent production steps is prohibited because of physical or operational constraints. This feature is typical of many industry-relevant production environments (HALL; SRISKANDARAJAH, 1996; MOCCELLIN *et al.*, 2018). In the notation by Lawler *et al.* (1993) , the problem is defined as: $O_m|block|C_{max}$. However, variants of the OSSP taking into account machine blocking and makespan minimization as performance measures have not been previously reported in the literature over the last four decades of research on the problem (AHMADIAN *et al.*, 2021b).

The significant contributions of blocking constraints with the open shop problem are: Lin, Lee and Pan (2008) presented a multi-stage processing OSSP with dedicated transportable machines with blocking constraints. The objective function is the total completion time minimization for all processing stages. Naderi, Najafi and Yazdani (2012) presented an OSSP without intermediate buffer, considering multiple machines and with the objective function of total tardiness minimization. Finally, Naderi and Zandieh (2014) studied an OSSP in an environment with machine blocking, considering multiple machines and jobs and proposing three MILP models and two metaheuristics (variable neighborhood search and genetic algorithm). The objective function was makespan minimization.

None of the cited papers proposed a formal definition for the open shop problem with blocking constraints, the definition of properties as lower bounds, and neither an integrated approach of exact and approximate methods for the problem's solution.

Therefore, these points constitute a gap to be filled in the present research project dissertation. The present doctoral dissertation investigates the open shop scheduling problem with machine blocking constraints and makespan minimization. A two-stage constrained scheduling model will be applied to solve the problem, hybridizing the CP models of the classic open shop and the variant open shop with blocking. Furthermore, new sets of challenging instances will be introduced to test the proposed solution methods with benchmarking methods from the current OSSP literature.

4.4 Open shop with operations repetitions (OSSPRM)

Timkovsky (2004) addressed a class of production scheduling problems, called production cycle scheduling problems, in which some jobs can be reprocessed on some machines with a certain number of repetitions. For example, this situation occurs in microchip manufacturing in a VLSI technology environment. In the open shop scheduling problem with repetitions (OSSPR), jobs can be processed on any machine more than once. In this problem, all jobs can be scheduled without constraints, substantially increasing the number of feasible solutions compared to the OSSP. In the notation by Lawler *et al.* (1993), the problem is defined as: $O_m|rcrc|C_{max}$.

A practical application example of the proposed variant can be described: a large automotive garage for vehicle maintenance and repair. Each workstation (box) has a mechanic with a set of tools. Consequently, each workstation can receive any type of vehicle and perform all necessary operations. Given the heterogeneity of the workstations, expressed in terms of technical ability of mechanics and the characteristics of the available tools, each workstation can perform the jobs with different processing times. The set of vehicles are the jobs to be processed at a set of workstations. These workstations are machines and process a set of maintenance activities on vehicles. The objective of the problem is to find a sequence of processing operations minimizing the scheduling duration (makespan).

This feature addresses a new open shop environment in which a given job can be processed, for different operations, more than once for the same machine. In the recent literature search on the open shop problem (ANAND; PANNEERSELVAM, 2016; ADAK; AKAN; BULKAN, 2020; AHMADIAN *et al.*, 2021a), there is no mention of the variant proposed in this doctoral dissertation, and is therefore unpublished. Moreover, Ahmadian *et al.* (2021a) in its review of the last four decades of OSSP publications, proposes the study of open shop considering the reprocessing of the jobs as an fundamental contribution to be made. Hence, this is one of the gaps that the project tries to address.

In recent years, new variants of the open shop have been proposed. Roshanaei, Esfehani and Zandieh (2010) presented an open shop scheduling problem with sequence-dependent setup times. Bai and Tang (2013) proposed an open shop problem considering makespan minimization and job release dates. Naderi and Zandieh (2014) studied a no-wait open store problem, in which there are no intermediate buffers between machines. In addition, Bai and Tang (2013) presented a flexible open shop, in which the scheduling concurrently considers the processing and routing of jobs through the production steps. Mosheiov *et al.* (2018) and Sheikhalishahi *et al.* (2019) addressed open shop environments taking maintenance issues into account. Finally, Aghighi *et al.* (2021) presented an open shop environment with reverse flows.

The problem considers a variant not yet reported in the literature that has important applications in the industrial and service sectors, especially in maintenance management activities, in which repetition of activities may occur. In the project, new properties, data sets, and methods will be proposed to solve the problem from an exact approach, such as linear and constraint programming models and through metaheuristics, such as variable neighborhood search (VNS) using the search strategy with constraint programming (CP) to provide better solutions with an admissible computational effort.

4.5 Open shop with vehicle routing (OSSP-VRP)

The integration between production and distribution in operations management has become an essential factor for the operational success of industries. A large part of the product costs involves the production and transportation phases along the supply chain, with logistics costs almost 30 percent of the total cost. Moreover, the strategy of solving these problems in an integrated solution is superior to solving them in a separate and sequential solution (DARVISH; COELHO, 2018).

Therefore, aiming for ways to integrate the scheduling of production and distribution of products becomes critical to the developments of modern companies. Through integrated scheduling models, jobs are first processed by machines and then delivered to customers by vehicles.

The problem by definition is to schedule a set of jobs j to be processed on a set of machines m, where each machine and job operation has a processing time. When all the operations required for a job are finished, it goes to an intermediate buffer. All jobs are processed and located in a single depot. Each job has a specific volume and must be delivered to a customer via a single vehicle with finite capacity. The travel time from depot to job customers and one customer to another is defined by a matrix of travel times. Each route to deliver the jobs must not exceed the vehicle's maximum capacity and must start and end at the depot. The objective of the problem is to minimize the duration of production scheduling plus delivery time (makespan) by finding a sequence for processing the operations and routing the orders to the customers. Since the problem is integration between scheduling and distribution, the classical notation of scheduling problems of Lawler *et al.* (1993) is not applicable.

Regarding integrated production and distribution problems considered in the present dissertation, the following researches can be mentioned: Ullrich (2013) proposed a parallel machine environment and distribution with time window constraints. The authors proposed a genetic algorithm to solve the problem. Farahani, Grunow and Günther (2012) developed a study for perishable products applying a MILP model, aiming to improve the quality of service, with the optimization of production integrated with the routes. Chang, Li and Chiang (2014) proposed an ant colony optimization algorithm (ACO) for solving the problem of parallel machines integrated with the distribution with multiple capacitated vehicles. The results showed the superiority of ACO over exact solving strategies. Finally, Tavares-Neto and Nagano (2019) developed an IGA for the parallel machine problem with sequence-dependent setup times, integrating the distribution through a single capacitated vehicle. The results showed the superiority of the IGA over exact MILP models.

No examples of the use of metaheuristics and exact methods for solving integrated scheduling and distribution problems in an open shop have been found in the literature (WANG; GRUNDER; MOUDNI, 2015; MOONS *et al.*, 2017). Therefore, the project dissertation proposes to study new metaheuristics, comparing their performances with exact strategies, such as MILP, and approximate ones, such as constructive heuristics, for the proposed OSSP-VRP problem. A Biased Random-Key Genetic Algorithm with Iterated Greedy Algorithm (BRKGA-IG) will be developed for the new problem OSSP-VRP with makespan minimization as the objective function.

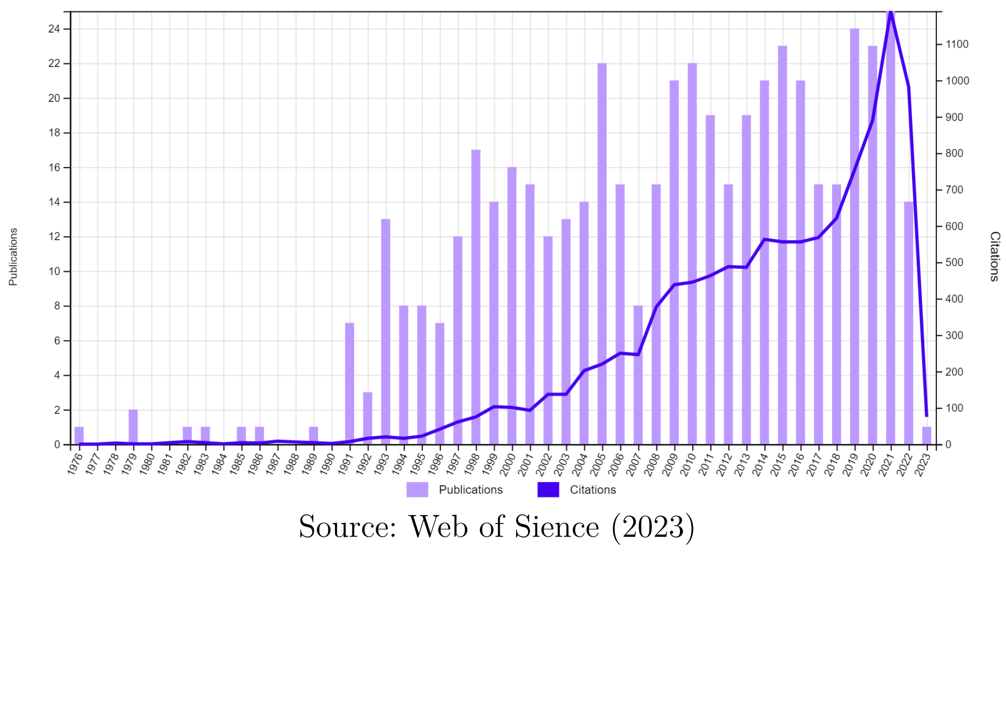
4.6 Integrative analyze of open shop problems

This section illustrates a bibliometric study, the summary of the open shop problems studied as chapters of the doctoral dissertation (articles), where is each open shop variant in the evolution of the project, the objectives of the project dissertation that each article undertakes to answer, and which solution methods are applied in each variant tested.

Based on the five presented open shop problems, new emerging variants of the problem were explored, considering aspects of setup times, machine blocking, repetition of operations, and vehicle routing. These variants are essential for improving the open shop scheduling problem literature, which has grown in recent years.

Figure 8 illustrates the evolution of citations and publications in the Web of Science database for the respective problem under analysis until February 2023 using in the search the keyword "open shop" AND schedul*. The citations of the area are growing, and still, have much space for exploration and research. Moreover, it has many practical applications that are part of a classic set of combinatorial optimization problems in production scheduling.

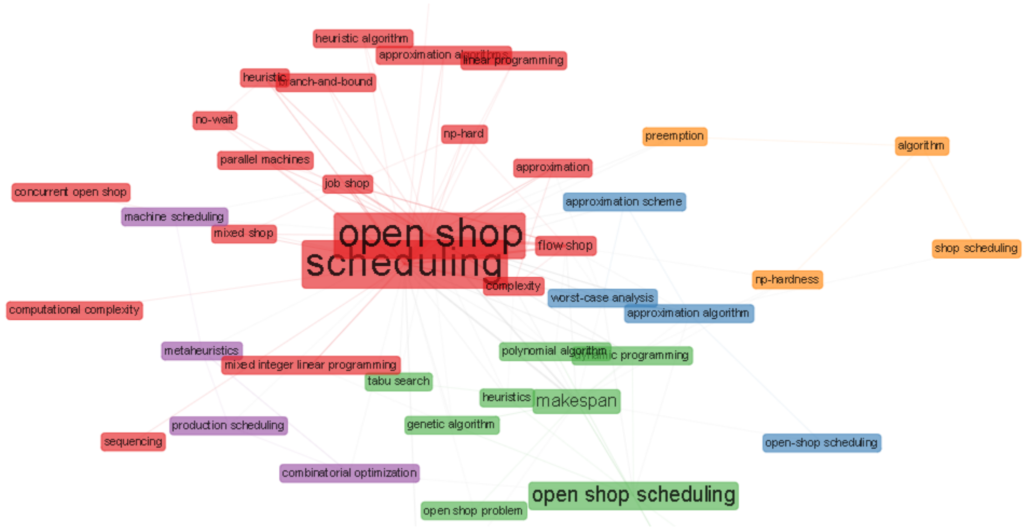
Figure 8 – Number of open shop articles published and citations per year



Source: Web of Sience (2023)

Another analysis concerns the main keywords of the articles and their co-occurrence relationships with each other. Figure 9 illustrates a co-occurrence network of the keywords in the same database used for the Figure 8, with articles from the open shop until February 2023. The size of the keywords is proportional to the occurrences in articles, and the distance of the relationship between keywords is proportionate to the number of occurrences they appear together in articles.

Figure 9 – Keyword co-occurrence network in open shop articles



Source: Bibliometrix (2023)

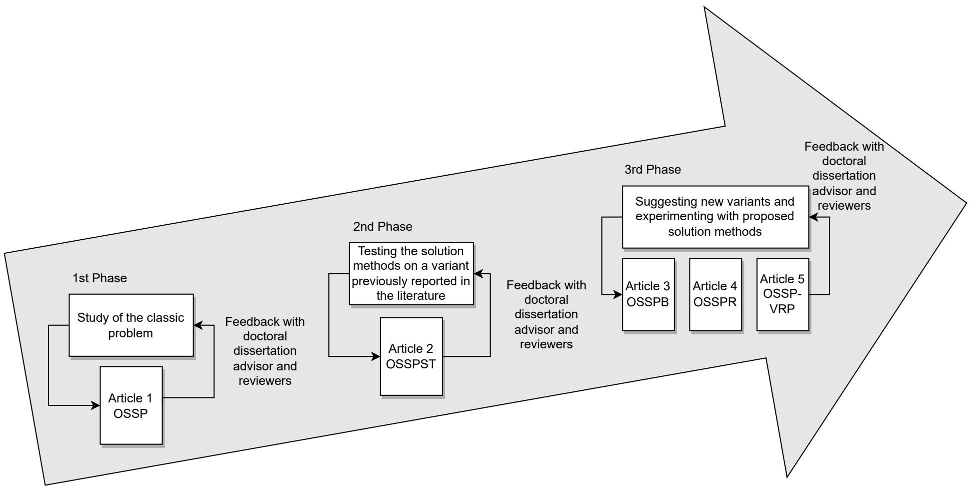
Figure 9 shows the frequent main terms in open shop articles, such as the makespan objective function, environment characteristics such as no wait, and problem-solving techniques such as genetic algorithms, tabu search, and MILP. No keywords related to the emerging variants proposed in the article were found, which is an opportunity to improve the literature, primarily studying these new variants of the open shop in the doctoral project.

The present doctoral project aims to develop in three main phases. Figure 10 illustrates each of three stages of the research with the articles to be developed in each phase. Each stage is developed through feedback with the doctoral dissertation advisor and article reviewers, better positioning the project, and correcting methodological problems of the constructs in the research.

The first phase of the research consists of studying the classical open shop problem to acquire knowledge about the main properties of the problem and modeling forms to be used in exact approaches and solution encoding forms to be used in heuristics and metaheuristics.

Furthermore, with this initial study, new efficient constructive heuristics will be proposed for the problem, based on new properties of the problem, such as idleness of

Figure 10 – Proposed stages of the article-based doctoral dissertation


1st Phase
Study of the classic problem
Article 1 OSSP
Feedback with doctoral dissertation advisor and reviewers
2nd Phase
Testing the solution methods on a variant previously reported in the literature
Article 2 OSSPST
Feedback with doctoral dissertation advisor and reviewers
3rd Phase
Suggesting new variants and experimenting with proposed solution methods
Article 3 OSSPB
Article 4 OSSPR
Article 5 OSSP-VRP
Feedback with doctoral dissertation advisor and reviewers

Source: Adapted from MIGUEL *et al.* (2010)

operations. These constructive heuristics will also be applied as initial solution methods for the new OSSP variants.

The second phase of the study tests new exact and meta-heuristic solution methods for OSSP variants proposed in the literature, such as OSSPST. These methods will have the constructive heuristics proposed in the previous phase as an initial solution.

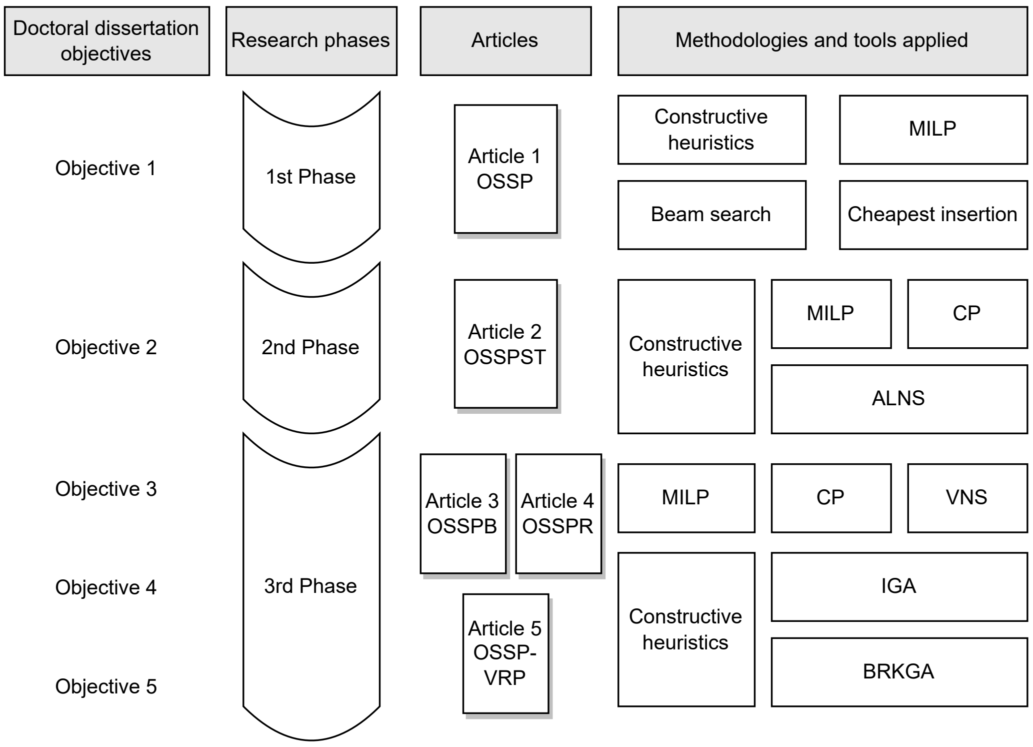
As the OSSPST already has a set of instances in the literature, the developed methods will be validated and compared with existing state-of-the-art solution methods, verifying the improvement in the quality of the problem solutions with setup times on already known instances.

With the properties of the classical problem known and new solution methods created for existing problems, there is an opportunity for proposing new variants for the OSSP. Finally, the third phase of the research consists of developing new variants for the open shop.

The third phase proposes the formal definition of the new problems, MILP and CP modeling, studies of new properties such as lower bounds formulation, creation of sets of instances, and experimentation comparing the proposed solution methods for the problems with metaheuristics implemented for similar problems and adapted for the new variants.

In summary, Figure 11 illustrates the integration between the doctoral dissertation objectives, the research phases, the articles, and the methodologies and tools used in each article. In addition, Figure 11 shows the importance of each article and what methods were used to achieve the main objectives of the doctoral dissertation.

Figure 11 – Integrative Review of proposed article-based doctoral dissertation



Source: Adapted from MIGUEL *et al.* (2010)

Analyzing Figure 11, it is possible to see the integration of each objective with each proposed article and the solving methods to be applied in each article. Mathematical and constraint programming models, constructive heuristics, and metaheuristics will solve the new variants, OSSP and OSSPST.

The following five chapters illustrate each of the proposed articles in the sequence of the objectives and phases of the doctoral dissertation. These are the classical problem and variants with setup times, machine blocking constraints, repetition of operations, and order delivery by vehicle routing.

5 NEW EFFICIENT HEURISTICS FOR SCHEDULING OPEN SHOPS WITH MAKESPAN MINIMIZATION

5.1 Introduction

Shop scheduling problems are widely studied optimization problems because of their many industrial applications. In the last decades, variants of the flow shop scheduling problem and job shop scheduling problem have received a lot of attention by researchers (see e.g. Fernandez-Viagas, Ruiz and Framinan (2017), Fan *et al.* (2018), Zhang, Wang and Xing (2019) for recent reviews on permutation flowshop, hybrid flowshop, and job shop scheduling, respectively). However, this has not been the same for the Open Shop Scheduling Problem (OSSP), which has received much less attention (ANAND; PANNEERSELVAM, 2016; ADAK; AKAN; BULKAN, 2020; AHMADIAN *et al.*, 2021a). This problem is first described by Gonzalez and Sahni (1976), and consists of scheduling a set of jobs on a set of machines, in which each job operation has an associated processing time. However, unlike flow shop and job shop scheduling problems, in the OSSP there are no predefined routes for the jobs in the machines. The OSSP has several industrial applications such as plastic molding, chemical processes, oil industry, and food production, while in the service sector, it is used to model medical care services, vehicle maintenance, telecommunications, and museum visit schedules (GONZALEZ; SAHNI, 1976; LIN; LEE; PAN, 2008; NADERI *et al.*, 2010; NADERI; NAJAFI; YAZDANI, 2012; VINCENT; LIN; CHOU, 2010; ABREU *et al.*, 2021; ABREU; TAVARES-NETO; NAGANO, 2021).

When the objective considered is the minimization of the maximum completion times of the jobs (makespan) and there is only one machine, the OSSP can be reduced to a single machine problem and every schedule is optimal. For the case of two machines, there are polynomial algorithms with optimality proof (GONZALEZ; SAHNI, 1976; PINEDO, 2016). However, for problems with three or more machines, the OSSP with makespan objective is NP-Complete (GAREY; JOHNSON, 2012). Therefore, although some branch-and-bound algorithms have been proposed for this problem (BRUCKER *et al.*, 1997; GUÉRET; PRINS, 1999; GUÉRET; JUSSIEN; PRINS, 2000), exact methods are quite limited for solving realistic-size problem instances.

In view of the aforementioned hardness of the OSSP with makespan objective, different approximate algorithms have been proposed. These can be broadly classified as either constructive heuristics, or local search/metaheuristic approaches. Regarding constructive heuristics, several contributions have been presented: Pinedo (2016) proposed two dispatching rules: Longest Alternate Processing Times (LAPT) and Longest Total Remaining Processing Times on Other Machines first (LTRPOM). LAPT schedules first the jobs with the longest processing time in other machine and the LTRPOM allocates a job first with the greater sum of processing times in other machine. Liaw (1998) presented a dispatching rule called Dense Schedule/Longest Total Remaining Processing (DS/LTRP), which is an improvement of the LTRPOM applying the well-known label correction algorithm (SKRIVER; ANDERSEN, 2000).

Ramudhin and Marier (1996) adapted to the OSSP the shifting bottleneck procedure heuristic, originally used to solve the job shop scheduling problem. The heuristic iteratively attempts to select the bottleneck job or machine to re-optimize the jobs' processing sequence. Strusevich (1998) proposed a greedy heuristic for the open shop, considering job priorities. The results for the three-machine case showed that the method obtains solutions with a maximum deviation of $\frac{3}{2}$ from the optimal solution. Guéret and Prins (1998) presented two constructive heuristics, the first based on dispatching rules and the second based on the construction of matchings in a bipartite graph. Bai and Tang (2011) proposed a modified rotation scheduling heuristic for the problem, with relevant theoretical contributions, such as proof of optimality when the number of jobs tends to infinity.

Regarding the application of local search methods for the OSSP, Colak and Agarwal (2005) proposed a neural network algorithm that uses ten heuristic rules and local search procedures, González-Rodríguez *et al.* (2010) proposed a heuristic local search with neighborhood procedure based on graph theory to solve OSSP with triangular fuzzy processing times. Finally, Naderi *et al.* (2010) presented new efficient constructive algorithms with a local search that outperforms other existing algorithms such as LAPT.

For the OSSP it is clear that, since the space of solutions is extremely large due to the absence of a predefined routing of the jobs, the solution encoding scheme plays a key role (AHMADIAN *et al.*, 2021a). Three different encoding schemes have been used in the literature, i.e.: the disjunctive graph representation, the rank matrix, and the permutation list. The disjunctive graph representation can be used exclusively for the makespan objective, and it was introduced by Liaw in a series of papers (see Liaw (1999) and Liaw (2000)). The rank matrix encoding was first proposed by Bräsel, Tautenhahn and Werner (1993), and it consists of a matrix in a data structure in which each row represents the sequence of operations for a given job on the machines, and each column represents the sequence of jobs on each machine. The permutation list encoding consists of a sequence of the operations (i.e. each tuple job, machine is given a number so a solution is represented by a sequence). Clearly, the permutation encoding scheme is much simpler, however its main disadvantage is its redundancy, as different sequences may indeed represent the same schedule. Naderi *et al.* (2010) presented four theorems to drastically reduce the redundancies in the permutation list encoding. They also propose four local search algorithms (IRH1, IRH2, IRH3 and IRH4) using these properties.

With respect to the application of metaheuristics for the OSSP, the aforementioned references by Liaw presented a tabu search (LIAW, 1999), a simulated annealing (LIAW, 1999), and a genetic algorithm (LIAW, 2000). Prins (2000) proposed a Genetic Algorithm (GA) using the permutation list encoding with two special features: a population with individuals with different makespan values and a procedure for reordering the generated chromosomes. This algorithm outperformed the then-existing heuristics and metaheuristics. Blum (2005) proposed a hybridized beam-search algorithm with ACO (Ant Colony Optimisation) using the permutation list encoding. Sha and Hsu (2008) presented a new Particle Swarm Optimization (PSO) algorithm using the permutation list scheme with an innovative encoding for the particles and a particle movement based on an insertion operator. Their computational results include several new best known solutions for the unsolved problems, and it is shown to outperform the algorithms by Liaw (1999), Prins (2000), and Blum (2005).

Also using the permutation list encoding, Ahmadizar and Farahani (2012) proposed a hybrid genetic algorithm (HGA) with a local search optimization procedure which outperforms the previously reported metaheuristics for the OSSP. Ghosn, Drouby and Harmanani (2016) proposed a parallel genetic algorithm (PGA) using deterministic and random moves. Pongchairerks and Kachitvichyanukul (2016) proposed a two level PSO with competitive performance on the benchmark instances. Finally, an extended genetic algorithm (EGA) was proposed by Hosseinabadi *et al.* (2018) to solve OSSP.

The Table 2 illustrates the main contributions of the literature to the classic OSSP. The authors, year of publication, characteristics of the problem such as the type of processing times, solution methods and main research contributions are illustrated.

Table 2 – Summary of the main contributions from the OSSP literature considering heuristic and metaheuristics approach.

Author	**Problem characteristics**	**Solution method**	**Contribution**
Colak and Agarwal (2005)	Classic OSSP	Neural network algorithm	Eficient ten heuristic rules and local search procedures with competitive results.
González-Rodríguez *et al.* (2010)	Triangular fuzzy processing times	Heuristic local search	Prove that feasibility and asymptotic convergence and proposed new benchmarking instances.
Naderi *et al.* (2010)	Classic OSSP	Constructive algorithms with local search	Theorems do reduce redundancies of the permutation list encoding and outperforms algorithms like LAPT.
Bai and Tang (2011)	Classic OSSP	Modified rotation scheduling heuristic	Proof of optimally when jobs trends to infinite.
Sha and Hsu (2008)	Classic OSSP	PSO	Innovative encoding of solutions and the results found several new best known solution for instances of literature.
Ahmadizar and Farahani (2012)	Classic OSSP	HGA	Outperforms the previously reported metaheuristics for the classic OSSP.
Ghosn, Drouby and Harmanani (2016)	Classic OSSP	PGA	Proposed new deterministic and random moves and got competitive results in Taillard instances.
Hosseinabadi *et al.* (2018)	Classic OSSP	EGA	New genetic operators and EGA outperforms all other tested methods.

Source: Authors.

As it can be seen from the above review, there are several methods to provide approximate solutions for the OSSP with makespan objective. However, we think that there is room for improving the state of the art of the problem by proposing new efficient constructive heuristics which incorporate some knowledge of the problem domain. More specifically, in this paper we first suggest using a look-ahead mechanism to estimate the contribution to the makespan of a partial solution in order to discard less-promising solutions, as well as some reasoning about the machines idle-time between operations. We believe that the development of efficient constructive heuristics for the OSSP is important for (at least) two reasons: 1) efficient constructive heuristics may provide high quality solutions in reduced computation times, which is required in many manufacturing environments and that allows to tackle problems of realistic size –particularly for the OSSP as the space of solutions grows very quickly with the problem size–, and 2) most metaheuristics and local search techniques use some constructive heuristic(s) as a starting solution, so designing more efficient constructive heuristics also boosts the performance of these procedures. These two aspects will be checked when developing our proposals. Once these fast constructive heuristics are developed, we combine them with a beam search algorithm and a cheapest insertion procedure. Finally, these are embedded in a local search (LS) procedure which uses the permutation list encoding but takes into consideration the four redundancy theorems proposed by Naderi *et al.* (2010) to overcome the disadvantages of the chosen encoding. All the algorithms proposed in the paper are compared with the existing constructive heuristics and local search approaches (i.e. LAPT, LTRPOM, DS/LTRP, EGA, IRH1, IRH2, IRH3 and IRH4) in an exhaustive computational experience.

The remainder of this paper is organized as follows: in Section 2, the scheduling problem treated in this paper is formally stated and a Mixed-Integer Linear Programming (MILP) model that will be used to obtain the optimal solutions for small-sized instances is presented. In Section 3, the proposed algorithms are described; in Section 4, we discuss some results of the computational experiments and statistical tests. Finally, in Section 5 we describe some conclusions and suggestions for future works.

5.2 Problem statement and MILP model

The problem considers n jobs that must be processed in m machines. Each job has a processing time on each machine and can visit the machines in any order. Furthermore, the usual hypotheses in scheduling apply: The processing of operations on the machines occurs at different times, i.e., a particular job can not be processed at the same time on more than one machine. In addition, we deal with the non-preemptive case of the OSSP, hence the processing of the jobs cannot be interrupted, i.e., the job once started on a machine, it must be processed until the end of the task. The objective of the decision problem is to minimize the maximum completion time among the jobs (makespan). In the notation of Lawler *et al.* (1993) , the problem is defined as: $O_m||C_{max}$.

If we use the permutation list as an encoding scheme (see e.g. Khuri and Miryala (1999)), a solution of the problem is given by a sequence s containing all the operations to be performed in the shop. The schedule corresponding to solution s consist in scheduling operation k corresponding to job j on machine i in order to start as earliest as possible but not before any previous job in the schedule. A pseudo-code of this active scheduler decoding scheme is given in Figure 12.

As an example, we present the classic instance GP03-01 of Guéret and Prins (1999) in Table 3. The instance has three jobs and three machines. Using the permutation list encoding, the operations to be performed in an instance with 3 jobs and 3 machines are presented in Table 4, where O_{ij} is the operation of job j in machine i. Taking into account also the processing times in Table 3 (as processing time p_{ij} of job j in machine i), it can be seen that the sequence $s = (9, 3, 5, 6, 4, 8, 7, 2, 1)$ returns a solution with a makespan of 2064 time units, as shown in Figure 13.

Table 3 – Processing times for open shop example

p_{ij} (O_{ij})	J_1	J_2	J_3
M_1	661 (O_{11})	168 (O_{12})	171 (O_{13})
M_2	70 (O_{21})	489 (O_{22})	505 (O_{23})
M_3	333 (O_{31})	343 (O_{32})	324 (O_{33})

Source: Authors.

Table 4 – Operations for the presented instance.

Operation	1 (O_{11})	2 (O_{12})	3 (O_{13})	4 (O_{21})	5 (O_{22})	6 (O_{23})	7 (O_{31})	8 (O_{32})	9 (O_{33})
Machine	M_1	M_1	M_1	M_2	M_2	M_2	M_3	M_3	M_3
Job	J_1	J_2	J_3	J_1	J_2	J_3	J_1	J_2	J_3

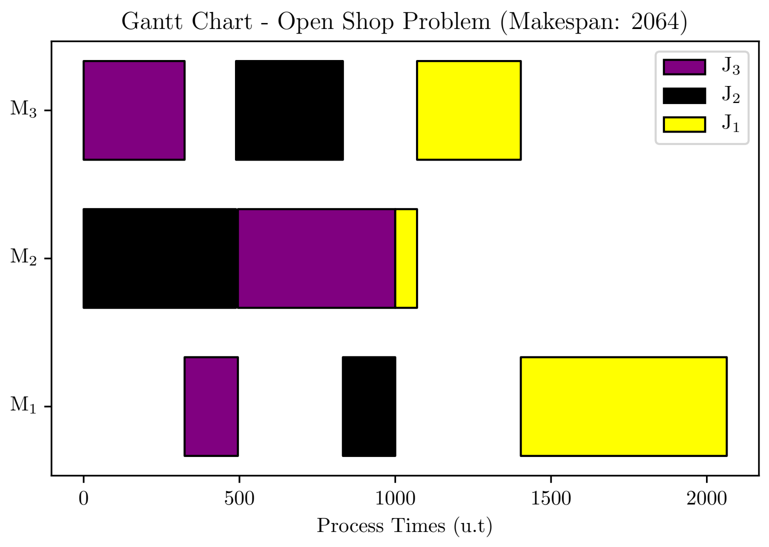
Source: Authors.

```
  Data: A solution with sequence $\Pi$
  Result: The maximum completion time ($makespan$)
1 $U \leftarrow \Pi$;
2 $M \leftarrow$ list with time accumulated in each machine;
3 $J \leftarrow$ list with time accumulated in each job;
4 while $\|U\| > 0$ do
5     $\pi_{zd} \leftarrow$ operation in the first position $\in U$;
6     $U \leftarrow U - \{\pi_{zd}\}$;
7     update $J$ and $M$ with time of $\pi_{zd}$ operation;
8 end
9 $makespan \leftarrow \max_{i \in \{1,\ldots,m\}} M_i$
```

Figure 12 – Active schedule decoding scheme procedure.

There are several ways to model the OSSP problem using mathematical programming, being different regarding to the way in which the decision variables are defined. More specifically, three types of notations can be used, i.e. positional notation, sequential

Figure 13 – Gantt chart for the presented solution.



Source: Authors

notation and time-indexed notation. In the study by Naderi *et al.* (2011b) it is illustrated that models with sequential notation perform better in OSSP problems due to their smaller number of variables and constraints, as compared to positional and time-indexed notation. Therefore, we have used the sequential notation in the MILP model developed.

Although MILP models are not efficient to solve medium and large size instances of many production scheduling problems due to their NP-hard nature, we find useful to present a MILP model for the problem with the aim of assessing in Section 5.4 the quality of the constructive heuristics proposed for small instances where the optima can be found. To do so, we adapt the formulation proposed by Naderi *et al.* (2011a) with sequential notation for the open shop with sequence-dependent setup times and total completion time minimization. We consider a dummy job 0 preceding the first job on each machine. Hereafter, the notation used for the problem is presented.

Indices and sets:

j: index for jobs $\{1,2,...,n\}$.

k: index for jobs (including the dummy job 0) $\{0,1,2,...,n\}$.

i, l: indices for machines $\{1,2,...,m\}$.

Parameters:

p_{ji}: processing time of job j on machine i (operation O_{ij}).

M: a large and positive number.

Decision variables:

C_{ji}: completion time of job j on machine i.

C_{max}: makespan.

Y_{jik}: 1 if operation O_{ij} is processed immediately after O_{ik}, and 0 otherwise.

X_{jil}: 1 if operation O_{ij} is processed after O_{lj}, and 0 otherwise.

The proposed MILP model is as follows.

minimize

$$C_{max} \tag{5.1}$$

subject to

$$\sum_{k=0,k\neq j}^{n} Y_{jik} = 1, \qquad \forall j, i \tag{5.2}$$

$$\sum_{j=1,j\neq k}^{n} Y_{jik} \leq 1, \qquad \forall i, k > 0 \tag{5.3}$$

$$\sum_{j=1,j\neq k}^{n} Y_{ji0} = 1, \qquad \forall i \tag{5.4}$$

$$Y_{jik} + Y_{kij} \leq 1, \qquad \forall i, j < n, k > j \tag{5.5}$$

$$C_{ji} \geq C_{ki} + p_{ji} - (1 - Y_{jik}) \times M, \qquad \forall j, i, k, k \neq j \tag{5.6}$$

$$C_{ji} \geq C_{jl} + p_{ji} - (1 - X_{jil}) \times M, \qquad \forall j, i < m, l > i \tag{5.7}$$

$$C_{jl} \geq C_{ji} + p_{jl} - X_{jil} \times M, \qquad \forall j, i < m, l > i \tag{5.8}$$

$$C_{max} \geq C_{ji}, \qquad \forall j, i \tag{5.9}$$

$$C_{0i} = 0, \qquad \forall i \tag{5.10}$$

$$C_{ji} \in \mathbb{R}^{+}, \qquad \forall j, i \tag{5.11}$$

$$C_{max} \in \mathbb{R}^{+}, \tag{5.12}$$

$$Y_{jik} \in \{0, 1\}, \qquad \forall j, i, k \neq j \tag{5.13}$$

$$X_{jil} \in \{0, 1\}, \qquad \forall j, i < m, l > i \tag{5.14}$$

The objective function (5.1) is the minimization of the makespan. Set of constraints (5.2) ensures that all the jobs are scheduled only once on each machine. Set of constraints (5.3) enforces that each job present at most one successor on each machine. Set of constraints (5.4) guarantees that the dummy job is preceding any other job on each machine. Set of constraints (5.5) avoids that a given job is simultaneously the predecessor and successor of another job. Constraints sets (5.6), (5.7), and (5.8) guarantee that the jobs are processed in the machines according to the previously defined processing times. Set of constraints (5.9) determines the makespan (maximum among all completion times). Constraint set (5.10) determines the completion time of dummy job. Finally, constraint sets (5.11), (5.12), (5.13), and (5.14) determine the domain of the decision variables. The proposed model includes n^2m binary decision variables, $nm + 1$ continuous decision variables and

$nm\left(\frac{1}{2}+\frac{3}{2}n+m\right)+n$ constraints. The constraint with the largest size is the one found in equations (5.6) and (5.7) with the worst case complexity of $\mathcal{O}\left(mn^2+m^2n\right)$. Table 5 illustrates a comparison of number of constraints and decision variables of MILP, for several different instance sizes. The parameter m is the number of machines and n is the number of jobs.

Table 5 – Comparison of MILP formulation with examples of instances sizes for OSSP

Instances sets		MILP		
m	n	# integer variables	# continuous variables	# constraints
3	3	27	10	75
4	4	64	17	172
5	5	125	26	330
6	6	216	37	564
7	7	343	50	889
8	8	512	65	1320
9	9	729	82	1872
10	10	1000	101	2560
15	15	3375	226	8565
20	20	8000	401	20220

Source: Authors.

5.3 Proposed algorithms

This section is devoted to present the new algorithms proposed for the problem under consideration. More specifically, in Section 5.3.1 we present three constructive heuristics for the problem using a beam search and cheapest insertion procedure, while in Section 5.3.2 we present an efficient local search procedure with reduction of the search space which can be initialized with any of the aforementioned constructive heuristics. The constructive heuristics start from the initial solutions obtained by an adaptation to our problem of the methods by Abreu *et al.* (2020) for the problem with setups, and are combined with an efficient beam search strategy and cheapest insertion. Beam search algorithms have been successfully applied to other scheduling environments (RUIZ; STÜTZLE, 2008; DONG; HUANG; CHEN, 2008; KIZILAY *et al.*, 2019). However, to the best of our knowledge, beam search algorithms have not been tested in the open shop environment.

5.3.1 Constructive heuristics

In this section we present six constructive algorithms to solve the OSSP for makespan minimization. These heuristics adapt the algorithms proposed by Abreu *et al.* (2020) that presented high-quality results for the OSSP with sequence-dependent setup times to minimize the total completion time. In Section 5.3.1.1 we propose the Bounded

Insertion Constructive Heuristic + Beam Search (BICH-BS) algorithm, which uses a projection of the makespan of the complete sequence for each step of the construction procedure to select the most promising partial sequence. The rationale of this heuristic is to reduce the solution search space by discarding sequences that would increase the lower bound and, consequently, the makespan of the solution in the short term. In Section 5.3.1.2 we present Minimal Idleness Heuristic + Beam Search (MIH-BS), a new constructive procedure that takes into consideration the minimization of the idleness of the machines in the production environment under study. The rationale of this heuristic is the following: the insertion of operations with less (local) idleness provides an increase in the utilization of the machines and consequently, it can potentially reduce the makespan at the end of the solution. In Section 5.3.1.3 we propose a method combining the two above-mentioned strategies. All these constructive methods presented are embedded in a beam-search procedure to explore the potential of using the constructive heuristics as a starting point of a local search procedure. Finally, in Section 5.3.1.4 we propose a hybridization of three proposed constructive heuristic by Abreu *et al.* (2020) with adaptation of cheapest insertion as improvement heuristic for OSSP.

5.3.1.1 Bounded Insertion Constructive Heuristic + Beam Search (BICH-BS)

The BICH-BS algorithm that we proposed for the OSSP is the result of the hybridization of two general approximate procedures (i.e. BICH and BS) adapted for the problem under consideration with a new procedure for the search space reduction based on the machine released earlier. We first give a brief description of these procedures and how they have been hybridized, and secondly we provide the pseudo-code with a detailed explanation.

The BICH procedure was first proposed by Fernandez-Viagas and Framinan (2015b) for the permutation flowshop scheduling problem with makespan minimization (PFSP) subject to a maximum tardiness, and it can be considered a state-of-the-art algorithm for this problem. The BICH algorithm starts with an empty solution, and then, for each unscheduled operation, an estimation of the lower bound if the unscheduled operation is inserted is obtained. The operation with the best expected lower bound is selected and inserted in the current solution, and the algorithm continues constructing the solution until all operations have been inserted.

As it can be seen, the main idea of the BICH heuristic is to employ a mechanism to limit the number of solutions to be explored in the solution space, hence it seems particularly well-suited for combinatorial optimization problems with an extremely large number of feasible solutions. An adaptation of the BICH for the open-shop scheduling problem with sequence-dependent setups has been proposed by Abreu *et al.* (2020). Hence, for our problem we propose a procedure that uses this heuristic (where setup times are considered to be zero) as initial solution and then the so-obtained solution is improved using a beam search (BS) strategy. The BS is a search algorithm based on nodes search, very similar to Branch and Bound, but only the best β nodes are selected for expansion, thus consuming less computational time as only a subset of nodes from the set of all possible solutions to the problem are explored (BIRGIN; FERREIRA; RONCONI, 2020).

The main elements of both BICH and BS are adapted to our problem. For the BICH, the operation returning the lowest expected lower bound is selected. For the BS, for each iteration, several operations are considered to be inserted in the solution. The domain knowledge of the problem is the insertion of operations based on the expected lower bound of the problem. This hedge prevents placing operations in positions that contribute negatively to the expected lower bound.

The lower bound for the open shop required by BICH is calculated using the well-known Equation (5.15) (PINEDO, 2016):

$$LB = \max\left\{\max_{j\in\{1,\cdots,n\}} \sum_{i=1}^{m} p_{ij}, \max_{i\in\{1,\cdots,m\}} \sum_{j=1}^{n} p_{ij}\right\} \tag{5.15}$$

One can observe that this lower bound can be computed with low computational effort. In addition, the computation of the expected lower bound required for BS can be also performed in a fast manner. First, we must calculate the expected contribution to the makespan (EMC) with the addition of the operation of job j in machine k (π_{kj}) in a partial solution $\mathit{\Pi}$, using a matrix of processing times P where the processing times of the operations previously inserted in the partial solution $\mathit{\Pi}$ are equal to zero, and also that of the operation π_{kj} for the EMC calculation ($P_{kj} = 0$). Equation 5.16 presents EMC calculation.

$$EMC(\pi_{kj}, P) = \max\left\{\max_{l\in\{1,\cdots,n\}} \sum_{i=1}^{m} P_{il}, \max_{i\in\{1,\cdots,m\}} \sum_{l=1}^{n} P_{il}\right\}, \text{with } P_{kj} = 0 \tag{5.16}$$

Therefore, using ECM, the expected lower bound can be calculated with the makespan of the operations presented in the partial solution $\mathit{\Pi}$ with the operation π_{kj}, adding in the value of the makespan, the ECM considering the insertion of the operation π_{kj} in the partial solution. Finally, we present the expected lower bound of insertion of operation π_{kj} in the partial solution $\mathit{\Pi}$ in Equation (5.17).

$$makespan(\mathit{\Pi} \cup \{\pi_{kj}\}, p) + ECM(\pi_{kj}, P) \tag{5.17}$$

The parameter $\mathit{\Pi}$ is a (partial) sequence of operations, π_{kj} is the operation of job j in machine k, p is a default processing time matrix, and P is a processing time matrix with the processing time of the operation π_{kj} and all other allocated in $\mathit{\Pi}$ equal to zero.

The main feature regarding the hybridization of BS and BICH is the fact that the best β operations are tested concerning the expected lower bound in each iteration. However, in the classic BICH, a single solution is constructed for each iteration through the selection of the best operation; thereby, this algorithm does not evaluate solutions with the insertions of different operations. In contrast, in BICH-BS, the search tree adds the new nodes, and in the next iterations, the insertion of β operations in each of the generated nodes is tested. At the end of the algorithm, the so-built node with the lowest makespan is returned.

Note that, since forcing the BICH-BS algorithm to select the best β nodes from all the existing set may demand a high computational cost, we propose a new local search mechanism (LS) that performs an initial filter. More specifically, the filter chooses only operations that contain the machine that is released earliest. We considered the operation π_{kj} to calculate the expected lower bound, where k is the index of the machine released earlier. In this manner, the search is improved by reducing the number of operations whose expected lower bound has to be computed.

The complete pseudo code of the proposed algorithm BICH-BS is shown in Figure 14. In the pseudocode, EMC is a function that calculates the expected makespan contribution, with a processing times matrix P and considering the time of candidate operation π_{kj} equal to zero so the time of this operation in the partial lower bound is not considered. If BICH selects this operation, it is inserted in the solution, and its processing time in the P matrix will be equal to zero. Thus, this operation will not be considered in the next iterations. Finally, p is an example of an instance as presented in Table 3. The algorithm returns a sequence of operations $\Pi := \{\pi_{11}, \pi_{13}, ..., \pi_{mn}\}$ as a solution for the OSSP.

In lines 1-8 of the pseudo-code, the main parameters are initialized, including the $\mathcal{N}$ tree with the starting node with the empty parameters. $\mathcal{N}$ is a list of tuples where each tuple represents a node of a partial solution constructed in each iteration with the insertion of an operation into the sequence Π. N_i denotes a node of a partial solution with an index of i, z is a counter for the number of nodes created in each iteration of BS, and N_z is the node of the last partial solution created. Line 9 corresponds to the main loop of the algorithm, while all operations are not allocated to the last created node, the algorithm's steps must be executed. Lines 15-16 select the operations to be tested for the expected lower bound, consider only the possible operations to be programmed on the machine released earlier, with jobs still available for programming in this machine.

Lines 17-19 creates the new nodes for each candidate operation to be inserted into the solution in the current node of the for loop in line 11. The best node operation is inserted in the current node on lines 27-28. In lines 30-35, the best new nodes found are selected to be added to the $\mathcal{N}$ tree. Line 37 selects the best solution found from all the nodes that have been created by BICH-BS.

```
Data: EMC(.), $p$, $\beta$
Result: A sequence $\mathit{\Pi}_{best} := \{\pi_{11}, \pi_{12}, ..., \pi_{mn}\}$
1  $\mathit{\Pi} \leftarrow \{\}$;
2  $P \leftarrow$ copy($p$);
3  $M \leftarrow$ list with time cumulative in each machine;
4  $J \leftarrow$ list with time cumulative in each job;
5  $\Omega_k \leftarrow$ list with the jobs alocated in machine $k$, $\forall k \in \{1, ..., m\}$;
6  $\mathcal{N} \leftarrow$ set of nodes for solution, each node is a tuple;   // $\mathcal{N}_i \leftarrow (\mathit{\Pi}, M, J, \Omega, P)$.
7  $z \leftarrow 1$ ;   // number of nodes created in search.
8  $\mathcal{N} \leftarrow \mathcal{N} \cup (\mathit{\Pi}, M, J, \Omega, P)$
9  while $\|\mathcal{N}_z.\mathit{\Pi}\| < n \times m$ do
10     new_nodes $\leftarrow \{\}$;   // set of new nodes created.
11     foreach node $i \in \mathcal{N}$ do
12         if $\|\mathcal{N}_i.\mathit{\Pi}\| < n \times m$ then
13             continue search for next node, this node $i$ was completed;
14         end
15         machine $k \leftarrow \operatorname*{argmin}_{r \in \{1,...,m\}, \|\mathcal{N}_i.\Omega_r\| < n} M_r$;   // index from machine released more
            early.
16         $\mathcal{J} \leftarrow$ list of jobs $j$ sort by makespan($\mathcal{N}_i.\mathit{\Pi} \cup \{\pi_{kj}\}$, $p$) + $EMC(\pi_{kj}, P)$
            with $j \notin \mathcal{N}_i.\Omega_k$;
17         foreach $j \in \mathcal{J}$ do
18             if $j$ is the first job in the list $\mathcal{J}$ then
19                 w $\leftarrow j$;   // The node $\mathcal{N}_i$ continues the insertion with the best job $j$ by
                    BICH criteria.
20             else
21                 $\mathit{\Pi}', \Omega_k', P', P_{kj}' \leftarrow \mathcal{N}_i.\mathit{\Pi} \cup \{\pi_{kj}\}, \mathcal{N}_i.\Omega_k \cup \{j\}, copy(\mathcal{N}_i.P), 0$;
22                 $J', M' \leftarrow \mathcal{N}_i.J$, $\mathcal{N}_i.M$;
23                 update $J'$ and $M'$ with time of $\pi_{kj}$ operation;
24                 new_nodes $\leftarrow$ new_nodes $\cup$ $(\mathit{\Pi}', M', J', \Omega', P')$;
25             end
26         end
27         $\mathcal{N}_i.\mathit{\Pi}, \mathcal{N}_i.\Omega_k, \mathcal{N}_i.P_{kw} \leftarrow \mathcal{N}_i.\mathit{\Pi} \cup \{\pi_{kw}\}, \mathcal{N}_i.\Omega_k \cup \{w\}, 0$;
28         update $\mathcal{N}_i.J$ and $\mathcal{N}_i.M$ with time of $\pi_{kw}$ operation;
29     end
30     best_nodes $\leftarrow$ the $\beta$ best nodes $\in$ new_nodes by makespan;
31     foreach node $i \in$ best_nodes do
32         $z \leftarrow z + 1$;   // A new node is create
33         extract $\mathit{\Pi}$, $M$, $J$, $\Omega$, P from node $i$;
34         $\mathcal{N} \leftarrow \mathcal{N} \cup (\mathit{\Pi}, M, J, \Omega, P)$;
35     end
36 end
37 $\mathit{\Pi}_{best} \leftarrow$ the best solution found $\in \mathcal{N}_i.\mathit{\Pi}$ $\forall i \in \{1, ..., z\}$;
```

Figure 14 – Pseudocode of the BICH-BS heuristic

5.3.1.2 Minimal Idleness Heuristic + Beam Search (MIH-BS)

In this section we propose the MIH-BS algorithm, which is the results of hybridizing the MIH heuristic (ABREU *et al.*, 2020) for the open-shop problem with setups with the BS strategy. First we describe the MIH heuristic and then we describe its adaptation and hybridization with the BS.

MIH is a heuristic procedure which relies on the idea that classical dispatching rules for the OSSP are largely based on LPT algorithms to sort operations in descending order of their processing times. In view of the similarities of the open-shop with the parallel machine environment, the allocation of jobs with the longest processing times using LPT might be an interesting strategy. However, this may cause a high idle time when applied to the open shop: While in the parallel machine environment this idleness is zero, in the open shop it can increase the waiting time of a given solution. The idea of the MIH is that partial solutions with low values of cumulative processing times would usually present a low makespan in the final solution. Thus, an indicator for idleness can be calculated by the accumulated times for jobs and machines in the production system over the execution of operations in the scheduling sequence. If the cumulative time for a given job in the system is greater than the accumulated time for a given machine, it means that the machine will wait until the job is finished, and consequently, it can be allocated to the current machine. If the cumulative time of this job is lower than the cumulative time of a given machine, this job was already processed in another machine and its processing in the current machine will not result in idleness (ABREU *et al.*, 2020). Φ_{ij} represents the idleness generated by the allocation of job j to machine i. M and J store the cumulative processing times for each machine and job are stored in M_i and J_j respectively, and both are updated every time a new operation is inserted into the sequence. Equation (5.18) presents the procedure to calculate idleness.

$$\Phi_{ij} = \begin{cases} J_j - M_i, & \text{if } J_j > M_i \\ 0, & \text{otherwise} \end{cases} \tag{5.18}$$

The MIH algorithm starts with an empty solution, and all unscheduled operations are inserted and their idleness is computed. The operation that results in the lowest idleness is inserted, and the algorithm continues constructing the solution until all the operations have been inserted. In the case of a tie, the decision is arbitrary. When operations have the same idleness, for tie-breaking, the operation with the lowest index is selected.

For our problem, the MIH originally proposed for the open-shop with setups is adapted. The main difference with the BS hybridization presented before is that the best β operations are tested with respect to their idleness in each iteration. The new nodes are added to the search tree, and in the next iterations, the insertion of β operations in each of the generated nodes is tested. At the end of the algorithm, the node with the lowest makespan is returned.

Furthermore, for the MIH-BS algorithm selecting the best β nodes from all the existing sets may demand a high computational cost. To overcome this problem we suggest the same mechanisms of the BICH-BS algorithm in Section 5.3.1.1, i.e. the operations selected to calculate the expected idleness are only the operations present in the machine released earlier in order to optimize the search by reducing the number of operations in which the expected idleness has to be computed.

Figure 15 shows the complete pseudo code for MIH-BS. In the algorithm, Φ_{ij} is the idle of operation π_{ij}, and p is a example of instance like in Table 3. As it can be seen, the structure of the algorithm is similar to that of BICH-BS. The main differences refer to lines 15-16 (where the operations to be tested are selected based on the expected idleness, considering only the possible operations programmed in the machine released earlier), and line 37 where the best solution found among all nodes that have been created by MIH-BS is selected.

5.3.1.3 A combined approach + Beam Search (BICH-MIH-BS)

Framinan and Perez-Gonzalez (2017) present a constructive heuristic in which there is a look-ahead procedure for measuring the potential contribution of the candidate operations to the objective function and an estimation of the contribution to the objective function of the non-scheduled operations in the sequence solution. This look-ahead mechanism is the main feature in our combined approach, taking into account the makespan lower bound, the machine's idleness, as well as the BS scheme.

On the basis of such reasoning, we develop a constructive heuristic that adapts MIH and BICH taking into consideration the contribution of an operation for the makespan objective as well as the idleness indicator with a beam-search procedure. We adopt a weight aggregation function for combining the two objectives, i.e. idleness minimization and makespan minimization.

Let Ψ_{ij} be a performance indicator for the insertion of the operation π_{ij} in the permutation, α the weight of the expected contribution for the idleness minimization, Φ_{ij} the expected contribution for the idleness minimization, p is a default processing time matrix, and P is a processing time matrix with the processing time of the operation π_{ij} and all other allocated in Π equal to zero. The performance indicator Ψ_{ij} can be computed as follows:

$$\Psi_{ij} = (1-\alpha) \times (makespan(\Pi \cup \{\pi_{ij}\}, p) + EMC(\pi_{ij}, P)) + \alpha \times \Phi_{ij} \tag{5.19}$$

If $\alpha = 1$ the combined approach is equal to MIH and if $\alpha = 0$ the combined

```
Data: $\Phi$, $p$, $\beta$
Result: A sequence $\mathit{\Pi}_{best} := \{\pi_{11}, \pi_{12}, ..., \pi_{mn}\}$
1  $\mathit{\Pi} \leftarrow \{\}$;
2  $P \leftarrow$ copy($p$);
3  $M \leftarrow$ list with time cumulative in each machine;
4  $J \leftarrow$ list with time cumulative in each job;
5  $\Omega_k \leftarrow$ list with the jobs alocated in machine $k$, $\forall k \in \{1, ..., m\}$;
6  $\mathcal{N} \leftarrow$ set of nodes for solution, each node is a tuple;   // $\mathcal{N}_i \leftarrow (\mathit{\Pi}, M, J, \Omega, P)$.
7  $z \leftarrow 1$ ;   // number of nodes created in search.
8  $\mathcal{N} \leftarrow \mathcal{N} \cup (\mathit{\Pi}, M, J, \Omega, P)$
9  while $\|\mathcal{N}_z.\mathit{\Pi}\| < n \times m$ do
10     new_nodes $\leftarrow \{\}$;   // set of new nodes created.
11     foreach node $i \in \mathcal{N}$ do
12         if $\|\mathcal{N}_i.\mathit{\Pi}\| < n \times m$ then
13             continue search for next node, this node $i$ was completed;
14         end
15         machine $k \leftarrow \underset{r \in \{1,...,m\}, \|\mathcal{N}_i.\Omega_r\| < n}{\operatorname{argmin}} M_r$;   // index from machine released more early.
16         $\mathcal{J} \leftarrow$ list of jobs $j$ sort by $\Phi_{kj}$ with $j \notin \mathcal{N}_i.\Omega_k$;
17         foreach $j \in \mathcal{J}$ do
18             if $j$ is the first job in the list $\mathcal{J}$ then
19                 w $\leftarrow j$;   // The node $\mathcal{N}_i$ continues the insertion with the best job $j$ by MIH criteria.
20             else
21                 $\mathit{\Pi}', \Omega'_k, P', P'_{kj} \leftarrow \mathcal{N}_i.\mathit{\Pi} \cup \{\pi_{kj}\}, \mathcal{N}_i.\Omega_k \cup \{j\}, copy(\mathcal{N}_i.P), 0$;
22                 $J', M' \leftarrow \mathcal{N}_i.J,\ \mathcal{N}_i.M$;
23                 update $J'$ and $M'$ with time of $\pi_{kj}$ operation;
24                 new_nodes $\leftarrow$ new_nodes $\cup\ (\mathit{\Pi}', M', J', \Omega', P')$;
25             end
26         end
27         $\mathcal{N}_i.\mathit{\Pi}, \mathcal{N}_i.\Omega_k, \mathcal{N}_i.P_{kw} \leftarrow \mathcal{N}_i.\mathit{\Pi} \cup \{\pi_{kw}\}, \mathcal{N}_i.\Omega_k \cup \{w\}, 0$;
28         update $\mathcal{N}_i.J$ and $\mathcal{N}_i.M$ with time of $\pi_{kw}$ operation;
29     end
30     best_nodes $\leftarrow$ the $\beta$ best nodes $\in$ new_nodes by makespan;
31     foreach node $i \in$ best_nodes do
32         $z \leftarrow z + 1$;   // A new node is create
33         extract $\mathit{\Pi}$, $M$, $J$, $\Omega$, P from node $i$;
34         $\mathcal{N} \leftarrow \mathcal{N} \cup (\mathit{\Pi}, M, J, \Omega, P)$;
35     end
36 end
37 $\mathit{\Pi}_{best} \leftarrow$ the best solution found $\in \mathcal{N}_i.\mathit{\Pi}\ \forall i \in \{1, ..., z\}$;
```

Figure 15 – Pseudocode of the MIH-BS heuristic

approach is equal to BICH. This heuristic need finding the best empirical α to solve the proposed problem.

As it can be seen, the algorithm is similar to the ones presented in the Section 5.3.1.1 and 5.3.1.2, just changing the selection criteria to Eq. (5.19) in order to select the operation to be inserted in the solution.

Therefore, BICH-MIH starts with an empty solution, and all unscheduled operations are considered by calculating their hybrid indicator. The operation resulting in the smallest Ψ is inserted, and the algorithm continues the construction of the solution until all the operations have been inserted.

The main difference with the BS hybridization is that, in each iteration, the best β operations are tested with respect to the hybrid criterion in Eq. (5.19). The new nodes are added to the search tree and, in the next iterations, the insertion of β operations in each of the generated nodes is tested. At the end of the algorithm, the node with the lowest makespan is returned.

For the MIH-BS algorithm selecting the best β nodes from the entire existing set may require a high computational cost. To overcome this problem we use the same mechanisms of the BICH-BS and MIH-BS algorithms proposed in Section 5.3.1.1 and 5.3.1.2, respectively. The operations selected to calculate the hybrid indicator Ψ are only operations present in the machine released earlier to optimize the search by reducing the number of operations to be computed.

Figure 16 shows the complete pseudo code for BICH-MIH-BS. In the pseudo code, As discussed previously, $EMC(.)$ calculates the expected makespan contribution of instance, Ψ_{ij} is the indicator combining the expected makespan and the idleness of inserting operation π_{ij}, and p is a example of instance like in Table 3.

The algorithm is similar to BICH-BS and MIH-BS. The main differences are in lines 15-16, where the operations to be tested is selected based on the combined indicator of makespan and the expected idleness, considering only the possible operations to be programmed on the machine released earlier, with jobs still available for programming in this machine. Line 42 selects the best solution found among all the nodes that have been created by BICH-MIH-BS.

5.3.1.4 Constructive heuristics with Cheapest Insertion (IST)

The cheapest insertion heuristics is a constructive heuristic used in many production scheduling problems (see e.g. Wu and Che (2020), or Rossi and Nagano (2020)). It starts with a pre-established sequence of operations and it constructs a solution by inserting the unscheduled operations one by one in an iterative manner. Each operation is inserted in the position where it obtains the best value of the objective function (cheapest insertion).

```
Data: $EMC(.)$, $\Phi$, $\Psi$, $p$, $\beta$, $\alpha$
Result: A sequence $\mathit{\Pi}_{best} := \{\pi_{11}, \pi_{12}, ..., \pi_{mn}\}$
$\mathit{\Pi} \leftarrow \{\}$;
$P \leftarrow$ copy($p$);
$M \leftarrow$ list with time cumulative in each machine;
$J \leftarrow$ list with time cumulative in each job;
$\Omega_k \leftarrow$ list with the jobs alocated in machine $k$, $\forall k \in \{1, ..., m\}$;
$\mathcal{N} \leftarrow$ set of nodes for solution, each node is a tuple;    // $\mathcal{N}_i \leftarrow (\mathit{\Pi}, M, J, \Omega, P)$.
$z \leftarrow 1$ ;    // number of nodes created in search.
$\mathcal{N} \leftarrow \mathcal{N} \cup (\mathit{\Pi}, M, J, \Omega, P)$
while $\|\mathcal{N}_z.\mathit{\Pi}\| < n \times m$ do
    new_nodes $\leftarrow \{\}$;    // set of new nodes created.
    foreach node $i \in \mathcal{N}$ do
        if $\|\mathcal{N}_i.\mathit{\Pi}\| < n \times m$ then
            continue search for next node, this node $i$ was completed;
        end
        machine $k \leftarrow \underset{r \in \{1,...,m\}, \|\mathcal{N}_i.\Omega_r\| < n}{\text{argmin}} M_r$;    // index from machine released more early.
        $\mathcal{J} \leftarrow$ list of jobs $j$ sort by $\Psi_{kj}$ with $j \notin \mathcal{N}_i.\Omega_k$;
        foreach $j \in \mathcal{J}$ do
            if $j$ is the first job in the list $\mathcal{J}$ then
                w $\leftarrow j$;    // The node $\mathcal{N}_i$ continues the insertion with the best job $j$ by BICH-MIH criteria.
            else
                $\mathit{\Pi}', \Omega'_k, P', P'_{kj} \leftarrow \mathcal{N}_i.\mathit{\Pi} \cup \{\pi_{kj}\}, \mathcal{N}_i.\Omega_k \cup \{j\}, copy(\mathcal{N}_i.P), 0$;
                $J', M' \leftarrow \mathcal{N}_i.J$, $\mathcal{N}_i.M$;
                update $J'$ and $M'$ with time of $\pi_{kj}$ operation;
                new_nodes $\leftarrow$ new_nodes $\cup$ $(\mathit{\Pi}', M', J', \Omega', P')$;
            end
        end
        $\mathcal{N}_i.\mathit{\Pi}, \mathcal{N}_i.\Omega_k, \mathcal{N}_i.P_{kw} \leftarrow \mathcal{N}_i.\mathit{\Pi} \cup \{\pi_{kw}\}, \mathcal{N}_i.\Omega_k \cup \{w\}, 0$;
        update $\mathcal{N}_i.J$ and $\mathcal{N}_i.M$ with time of $\pi_{kw}$ operation;
    end
    best_nodes $\leftarrow$ the $\beta$ best nodes $\in$ new_nodes by makespan;
    foreach node $i \in$ best_nodes do
        $z \leftarrow z + 1$;    // A new node is create
        extract $\mathit{\Pi}$, $M$, $J$, $\Omega$, P from node $i$;
        $\mathcal{N} \leftarrow \mathcal{N} \cup (\mathit{\Pi}, M, J, \Omega, P)$;
    end
end
$\mathit{\Pi}_{best} \leftarrow$ the best solution found $\in \mathcal{N}_i.\mathit{\Pi}$ $\forall i \in \{1, ..., z\}$;
```

Figure 16 – Pseudocode of the BICH-MIH-BS heuristic

IST is a greedy constructive heuristic, being of simple implementation in the most diverse scheduling problems.

Here we propose hybridizing the BICH, MIH, and BICH-MIH described above with the improvement of the solutions through the IST heuristics, with the calculation of the makespan using a decoding scheme, explained in Figure 19.

More specifically, the algorithm starts with a list of operations sorted according to one of the three criteria: BICH, MIH, or BICH-MIH. Then, operations not yet allocated in the solution, present in the list of ordered operations, is tested in each possible positions in the solution. The chosen position is that where the best makespan is obtained. The algorithm finishes when all the operations have been inserted into the solution.

The complete pseudo-code of the cheapest insertion heuristics adapted for the OSSP is shown in Figure 17 where W is the list of operations sorted by some criteria and p is a sample instance as in Table 3. The Algorithm returns a solution $\mathit{\Pi}_{best} := \{\pi_{11}, \pi_{12}, ..., \pi_{mn}\}$.

```
   Data: $W$, $p$
   Result: A sequence $\mathit{\Pi}_{best} := \{\pi_{11}, \pi_{12}, ..., \pi_{mn}\}$
1  $\mathit{\Pi} \leftarrow \{\}$;
2  $W \leftarrow$ a solution ordered by a constructive heuristic as BICH, MIH or BICH-MIH;
3  best_make $\leftarrow$ best makespan generated in each interation;
4  best_pos $\leftarrow$ best position found by insertion $\pi ij$ in solution;
5  while $\|W\| > 0$ do
6      $\pi_{ij} \leftarrow$ first operation $\in W$;
7      best_make $\leftarrow \infty$
8      foreach position pos $\in \mathit{\Pi}$ do
9          $\mathit{\Pi}' \leftarrow$ a solution with insertion of $\pi_{ij}$ in position pos in $\mathit{\Pi}$;
10         if makespan($\mathit{\Pi}'$, $p$) $<$ best_make then
11             best_make $\leftarrow$ makespan($\mathit{\Pi}'$, $p$);
12             best_pos $\leftarrow$ pos;
13         end
14     end
15     $\mathit{\Pi} \leftarrow$ solution with insertion of of $\pi_{ij}$ in position best_pos;
16     $W \leftarrow W/\{\pi_{ij}\}$;
17 end
```

Figure 17 – Pseudocode of the cheapest insertion heuristic

Lines 1-4 set the parameters for the execution of the algorithm. Line 5 corresponds to the algorithm's main while loop: while the list of ordered operations is not empty, the algorithm runs. Lines 6-7 select the operation to be tested in the current iteration. In lines 8-14, the operation is tested in all possible positions in the $\mathit{\Pi}$ solution. In lines 15-16, the operation is inserted in the best position found.

5.3.2 Local search

As already mentioned, since the search space of the OSSP is very large, neighborhood search algorithms play a key role for finding high-quality solutions, mainly for large-sized instances. The local search applied is the well-known 2-opt algorithm, with a best improvement strategy. This local search procedure includes the search space reduction mechanisms based on the theorems proposed by Naderi *et al.* (2010) aiming to reduce the movements that generate redundant solutions.

The search consists in the pairwise exchange (swap) of a given operation and the rest, respecting the redundancy constraints. When all feasible exchanges are performed, there are two possibilities: (a) if there is no improvement, the next operation in the sequence is selected; (b) otherwise, the swap that generates the best makespan is selected. Thereafter, the search is restarted so the procedure stops when all the feasible swaps are evaluated.

In Figure 18 the proposed local search is presented. In this figure, makespan refers to the objective function, redundancy is a function that returns whether a swap between two operations is redundant or not based in Naderi *et al.* (2010) theorems, Π is a solution that will receive a local search and p is a example of instance such as illustrated in Table 3.

The solution decoding used for the algorithms with local search procedures is different from the previous one used in constructive algorithms. The main change is the consideration of a non-delay schedule in makespan calculation, which consists in the minimization of the idle time of machines. With this type of decoding, a given machine is not kept idle if there are still jobs to be processed, thus no machine is kept idle at a time when it could start processing other operations (SHA; HSU, 2008). The decoded solutions with non-delay have an equal or better makespan than solutions decoded without non-delay. With non-delay decoding, multiple permutations get the same makespan value, which can reduce the search space of the solutions, improving the efficiency of the local search algorithms (NADERI *et al.*, 2010). Therefore, this decoding scheme prioritizes the processing of the operations with the earliest starting time in each iteration of the makespan calculation. This decoding is not used in the iterations of constructive heuristics due to its high computational requirements.

This decoding always prioritizes scheduling first the operations with lower start time s_{ij}. Where there is more than one operation with the same start time, the operation π_{zd} with the earliest relative position is prioritized. Figure 19 presents the decoding scheme used in the proposed local search.

```
Data: redundancy(.), $\Pi$, $p$
Result: A new sequence $W$
1  $W \leftarrow$ an sequence $\Pi$ generated by construtive heuristics;
2  $BestMake \leftarrow makespan(W)$;
3  $r \leftarrow m \times n$;
4  improvemment $\leftarrow$ True;
5  while $r > 0$ do
6      if $improvemment = False$ then
7          $r^{--}$
8      end
9      improvemment $\leftarrow$ False;
10     $\pi_{ij} \leftarrow$ operation in $r$ position in $W$;
11     for $r2 = 1$ to $m \times n$ do
12         $\pi_{zd} \leftarrow$ operation in $r2$ position in $W$;
13         if $redundancy(\pi_{ij}, \pi_{zd}) = False$ then
14             $WW \leftarrow$ solution with swap between $\pi_{ij}$ and $\pi_{zd}$ operations;
15             if $makespan(WW, p) < BestMake$ then
16                 improvemment $\leftarrow$ True;
17                 $BestMake \leftarrow makespan(WW, p)$;
18                 $r3 \leftarrow r2$;
19             end
20         end
21         if $improvemment = True$ then
22             $\pi_{zd} \leftarrow$ operation in $r3$ position in $W$;
23             $W \leftarrow$ solution with swap between $\pi_{ij}$ and $\pi_{zd}$ operations; // the best
                swap founded in search
24             $r \leftarrow m \times n$;                                   // the search restarts
25         end
26     end
27 end
```

Figure 18 – Local search with reduction of search space.

5.4 Computational results

The proposed constructive heuristics are evaluated using the literature test problems proposed by Taillard (1993), Guéret and Prins (1999) and Brucker *et al.* (1997), which are the usual testbeds employed in OSSP. In the test problems by Guéret and Prins a fixed interval for the processing times is considered, with random values uniformly distributed between 1 and 1000, and a constant value for the lower bound equal to 1000. Different problem sizes are considered with $n, m \in \{3, 4, 5, 6, 7, 8, 9, 10\}$. For each class we have randomly generated 10 test instances, totaling 80 instances. Brucker et al. test problems are generated with random values between 1 and 500 uniformly distributed and 6 sets of problem sizes $n, m \in \{3, 4, 5, 6, 7, 8\}$, totaling 60 instances. Taillard test problems were generated with random values uniformly distributed between 1 and 100, without a lower

```
   Data: A solution with sequence $\Pi$
   Result: A sequence $S := \{\pi_{11}, \pi_{12}, ..., \pi_{mn}\}$ with encoding scheme
 1 $S \leftarrow \{\}$;
 2 $U \leftarrow \Pi$;
 3 $M \leftarrow$ list with time acumulated in each machine;
 4 $J \leftarrow$ list with time acumulated in each job;
 5 $s_{ij} \leftarrow$ start time for processing of operation $\pi_{ij}$
 6 while $\|U\| > 0$ do
 7     $y \leftarrow \min\left\{\max_{i \in \{1,...,m\}} M_i, \max_{k \in \{1,...,n\}} J_k\right\}$;
 8     $R = \{\pi_{ij} | s_{ij} = y, \pi_{ij} \in U\}$;
 9     $\pi_{zd} \leftarrow$ operation in earliest relative position $\in R$;
10     $U \leftarrow U - \{\pi_{zd}\}$;
11     $S \leftarrow S + \{\pi_{zd}\}$;
12     update $J$ and $M$ with time of $\pi_{zd}$ operation;
13     update $s_{ij}$ based on $J$ and $M$ times, $\forall i \in M, j \in N$;
14 end
```

Figure 19 – Non-delay schedule decoding scheme procedure.

bound constraint. Problem classes were considered according to the combination of the 6 sets of problem size $n, m \in \{4, 5, 7, 10, 15, 20\}$. For each size, 10 instances are randomly generated, totaling 60 instances. The complete instances set has 192 instances.

The above mentioned algorithms were implemented in the Intel Distribution for Python integrated development environment https://software.intel.com/content/www/us/en/develop/tools/distribution-for-python.html and were run in C with Cython library http://cython.org/ (BEHNEL *et al.*, 2011). The computational experience was performed on a PC with Intel Core i7-4771 CPU 3.50GHz and 12GB memory. The source codes, results of all computational tests, and statistical analyses are available at http://repositorio.uspdigital.usp.br/handle/item/447.

The parameter α of the proposed algorithm was tuned after several simulations. For each instance size, a value of α contained in the set $A = \{0.00, 0.11, 0.22, 0.33, 0.44, 0.56, 0.67, 0.78, 0.89, 1.00\}$ was tested. The best α values for BICH-MIH are presented in the Table 6 for each problem size. In the small-sized instances, the α value is close to 1 and the combined constructive heuristic allocates operations prevailing the reduction of idleness. In the large-sized instances, the α value is close to 0 and the constructive heuristic allocates operations prevailing bounded insertion. The parameter β is set as $\beta = 4$ after preliminary calibration experiments with $\beta = \{1, 2, ..., 10\}$, since this value was found to be a good trade-off between solution quality and computational times in all problem sizes tested.

The statistic used in the analysis of the computational experiments is the gap between the evaluated method (sol_{ik}) and best known solution (BKS_i), as presented in

Table 6 – The α values used for constructive heuristics BICH-MIH in each problem size.

Problem size	3	4	5	6	7	8	9	10	15	20
α	0.89	0.89	0.56	0.11	0.22	0.22	0.67	0.67	0.89	0.22

Source: Authors.

the Equation (5.20). The value sol_{ik} meaning the solution obtained by method k run on instance i, and BKS_i denotes the best known solution for instance i.

$$RPD_{ik} = \frac{sol_{ik} - BKS_i}{BKS_i} \cdot 100 \tag{5.20}$$

5.4.1 Computational results for constructive heuristics

Initially, we consider in our analysis constructive algorithms for the OSSP. The considered algorithms are listed below. All constructive algorithms used the decoding scheme procedure for the computation of the makespan of the final solution.

- Longest Processing Time (LPT): sort operations in non-increasing order of their processing times
- Shortest Processing Time (SPT): sort operations in non-decreasing order of their processing times.
- Longest Alternate Processing Times (LAPT): The priority rule developed by Pinedo (2016) for the OSSP.
- Longest total processing time (LTPT): a variant of the LAPT rule proposed by Naderi *et al.* (2010).
- Longest Total Remaining Processing Times on Other Machines first (LTRPOM): a priority rule developed by Pinedo (2016). It is a more general rule than LAPT.
- Dense Scheduling / Longest Total Remaining Processing (DS/LTRP) developed by Liaw (1998).
- Dense Scheduling / Longest Total Remaining Processing Time (DS/LTRPAM) developed by Colak and Agarwal (2005).
- Modified Rotation Scheduling (MRS) a constructive heuristic developed by Bai and Tang (2011).
- Cheap Insertion Heuristic with LPT initial solution (ISTH) adapted for OSSP.
- Bounded Insertion Constructive Heuristic (BICH) developed by Abreu *et al.* (2020).

- Minimal Idleness Heuristic (MIH) developed by Abreu *et al.* (2020).
- Combined algorithm approach (BICH-MIH) developed by Abreu *et al.* (2020).
- Bounded Insertion Constructive Heuristic with Beam Search procedure (BICH-BS).
- Minimal Idleness Heuristic with Beam Search procedure (MIH-BS).
- Combined algorithm with Beam Search procedure (BICH-MIH-BS).
- Bounded Insertion Constructive Heuristic with cheap Insertion procedure (BICH-IST).
- Minimal Idleness Heuristic with cheap Insertion procedure (MIH-IST).
- Combined algorithm with cheap Insertion procedure (BICH-MIH-IST).

For comparison purposes, we are also considering the results of the MILP model expressed by Equations (5.1)-(5.13), which was modeled and run on IBM ILOG CPLEX version 12.7, with 3600s of time limit.

A summary of the computational results is presented in Table 7. It can be highlighted that the computational times for the constructive heuristics (without the BS or IST procedure) are negligible (less than 1 second). The results of Average RPD (ARPD) in each set of instance of Guéret and Prins, Taillard and Brucker are presented in Figures 20, 21 and 22 respectively.

In order to validate the results, it is important to verify whether the previous differences in the RPD values are statistically significant. We apply an analysis of variance (ANOVA) (MONTGOMERY, 2017). The p-value is very close to zero. We can see in Figure 23 the ARPD boxplot for all constructive heuristics tested with HSD Tukey group ($\alpha = 0.05$) of the similar mean result. We can see that there are statistically significant differences between the ARPD values among the constructive heuristics proposed. The combined approach with IST and the constructive heuristics with BS gets the best results.

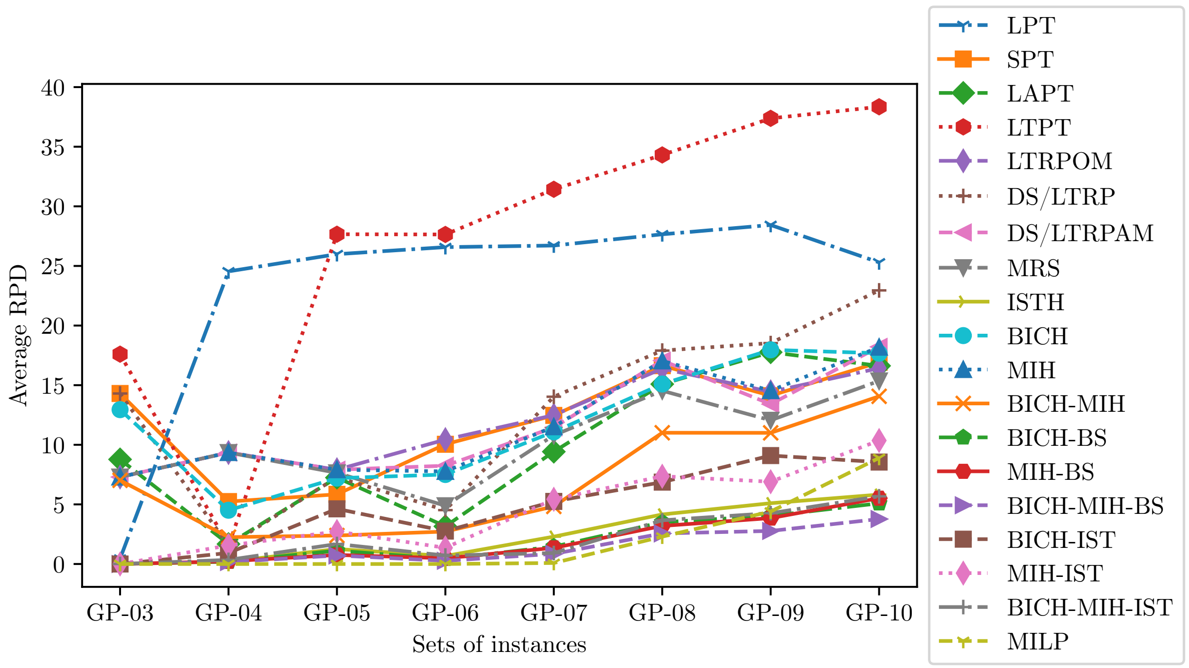
Table 7 – Results of constructive heuristics and mixed integer programming for each set of instances.

Benchmark	MILP	LPT	SPT	LAPT	LTPT	LTRPOM	DS/LTRP	DS/LTRPAM	MRS	ISTH	BICH	MIH	BICH-MIH	BICH-BS	MIH-BS	BICH-MIH-BS	BICH-IST	MIH-IST	BICH-MIH-IST
Guéret and Prins																			
GP-03	**0.00**	0.45	14.30	8.77	17.60	7.29	14.30	7.29	7.29	**0.00**	12.95	7.29	7.02	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**
GP-04	**0.00**	24.54	5.24	1.67	1.54	9.35	1.54	9.35	9.35	0.20	4.51	9.35	2.25	0.20	0.23	0.17	0.93	1.56	0.35
GP-05	**0.00**	25.99	5.84	7.24	27.66	7.91	7.27	7.91	7.65	1.26	7.20	7.91	2.39	1.12	0.76	0.71	4.63	2.73	1.68
GP-06	**0.00**	26.58	10.04	3.16	27.64	10.44	4.52	8.24	4.89	0.67	7.51	7.77	2.72	0.43	0.48	0.27	2.76	1.36	0.71
GP-07	**0.09**	26.71	12.46	9.42	31.43	12.48	14.03	11.45	10.80	2.30	11.09	11.51	4.83	1.46	1.35	0.85	5.25	5.43	0.92
GP-08	**2.29**	27.65	16.64	15.09	34.31	16.39	17.90	17.06	14.53	4.16	15.09	17.04	11.01	3.50	3.19	2.56	6.87	7.36	3.66
GP-09	4.40	28.43	14.12	17.76	37.38	14.44	18.53	13.41	12.05	5.10	17.96	14.54	11.00	4.03	3.85	**2.77**	9.09	6.90	4.26
GP-10	8.90	25.30	16.92	16.62	38.35	16.43	22.95	18.22	15.36	5.82	17.68	18.19	14.06	5.09	5.52	**3.77**	8.55	10.36	5.68
Taillard																			
tai_4x4	**0.00**	10.47	15.37	10.27	27.87	10.78	10.74	9.76	10.00	4.65	14.92	12.86	7.45	3.08	2.26	2.22	5.80	5.68	4.22
tai_5x5	**0.00**	14.93	14.07	14.42	27.91	15.12	19.37	16.79	10.05	7.41	16.06	14.31	10.42	3.38	3.30	3.01	8.33	9.89	6.06
tai_7x7	**1.28**	10.98	11.94	10.26	22.46	11.94	15.05	11.37	11.03	7.46	13.18	11.86	6.80	2.81	3.03	1.86	6.46	7.81	4.06
tai_10x10	3.44	7.06	8.95	8.77	14.02	7.01	17.24	6.34	7.12	5.14	9.17	6.96	5.48	1.48	1.01	**0.74**	5.27	4.74	2.63
tai_15x15	17.10	4.03	4.00	3.96	-	5.37	-	5.60	4.05	2.17	6.99	6.13	3.28	0.77	0.68	**0.33**	2.59	2.36	1.39
tai_20x20	38.25	1.99	3.52	4.19	8.11	2.77	18.15	1.98	2.83	1.67	4.06	3.86	1.57	0.53	0.44	**0.22**	1.51	2.05	0.61
Brucker																			
j3	**0.00**	5.80	15.77	13.93	17.05	15.68	15.68	7.66	5.75	3.56	19.10	10.94	4.01	2.63	2.63	2.63	12.10	6.81	4.12
j4	**0.00**	10.89	10.99	12.10	18.16	10.36	9.36	7.96	8.82	5.47	8.78	11.28	4.62	2.43	2.41	2.20	4.55	7.18	3.68
j5	**0.00**	14.19	14.57	16.52	28.31	16.24	13.64	13.86	9.31	5.36	17.23	13.87	8.28	1.86	3.16	1.86	10.33	8.53	5.91
j6	**0.00**	12.94	13.60	14.20	22.72	13.23	17.93	11.73	10.82	6.98	19.89	14.98	8.98	4.62	3.80	2.89	14.63	9.74	4.74
j7	**2.92**	10.13	13.56	13.56	35.51	15.26	14.79	15.09	10.20	8.69	14.42	17.37	12.68	4.07	5.06	3.91	12.25	9.31	5.72
j8	7.09	14.92	14.60	12.22	32.01	14.01	17.07	12.89	9.34	7.14	13.70	12.28	11.09	4.59	4.90	**4.07**	8.88	10.02	6.03
Min	**0.00**	0.45	3.52	1.67	1.54	2.77	1.54	1.98	2.83	**0.00**	4.06	3.86	1.57	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**
Average	4.29	15.20	11.83	10.71	24.74	11.62	14.21	10.70	9.06	4.26	12.57	11.51	7.00	2.40	2.40	**1.85**	6.54	5.99	3.32
Max	38.25	28.43	16.92	17.76	38.35	16.43	22.95	18.22	15.36	8.69	19.89	18.19	14.06	5.09	5.52	**4.07**	14.63	10.36	6.06

Source: Authors.

According to the results for the 80 Guéret and Prins test problems. The MILP model returns the best solutions for small and medium sizes classes of instances and BICH-MIH-BS returns the best solutions for large instances sizes with 9 and 10 problems size. The LPT and SPT rules are the worst algorithms for the instances analyzed. ISTH outperforms BICH-IST and MIH-IST, showing that the LPT initial sequence gives better results, but BICH-MIH-IST gets better results than ISTH. Therefore, the combined approach as the initial sequence indicates an improvement in the construction procedure of IST. The BS algorithms give the best results with ARPD less than 2%. Considering only the constructive heuristics, the proposed BICH-MIH-BS algorithm presented the best average results for all the eight classes of instances.

Figure 20 – Benchmark of constructive heuristics for each set of instances proposed by Guéret and Prins (1999).

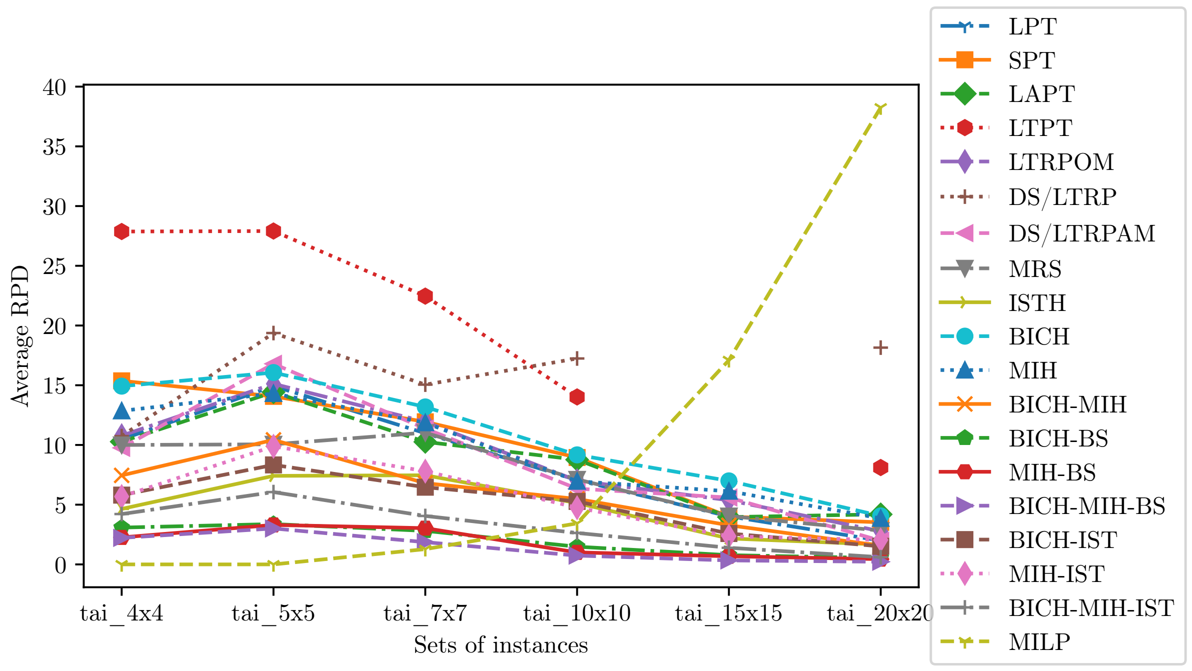


Source: Authors

With regard the results for the 60 Taillard test problems, the following comments can be made. The MILP model returns the best average results only for the 4, 5 and 7 problems sizes. In general, the LTPT, BICH and SPT rules are the worst algorithms for this set of problems. For large problems size, the MILP model returns the worst results. The behavior of all other constructive heuristics is similar to the results for the Guéret and Prins test problems. The combined BICH-MIH-BS presents the best results for the largest instances (tai_10x10, tai_15x15 and tai_20x20).

With respect to the results for the 80 Brucker et al. test problems, the following comments can be done. The MILP model finds the optimal solution within the time limit for the test instances with 4, 5, and 7 jobs. However, for instances with size 8 the MILP model

Figure 21 – Benchmark of constructive heuristics for each set of instances proposed by Taillard (1993).



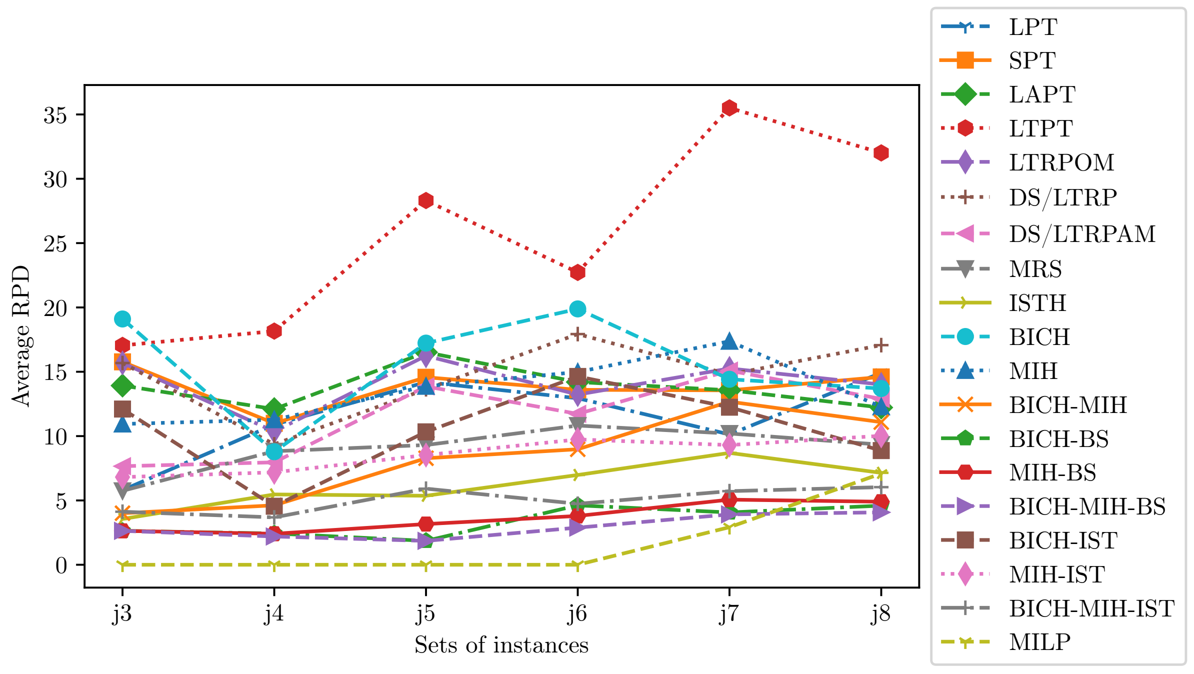
Source: Authors

returns solutions of average quality within the allowed time limit. The LTPT presents the worst results compared to all others methods. Considering only the constructive heuristics, the proposed BICH-MIH-BS algorithm presents the best average results for all the eight classes of instances. For largest instances, (size 8) the BICH-MIH-BS presents the best results.

Taking into consideration the three sets of instances, the proposed approach BICH-MIH-BS presents the lower ARPD for all the analyzed constructive heuristics as well as lower than the MILP model within the time limit. With respect to the ARPD, the difference among BICH-MIH-BS and MILP methods is significant because they are clustered in different groups (with h and fg letters, respectively). Also, they are represented in different color groups that indicate groups with different ARPD. Therefore, the BICH-MIH-BS outperforms MILP (within the given time limit) in terms of ARPD.

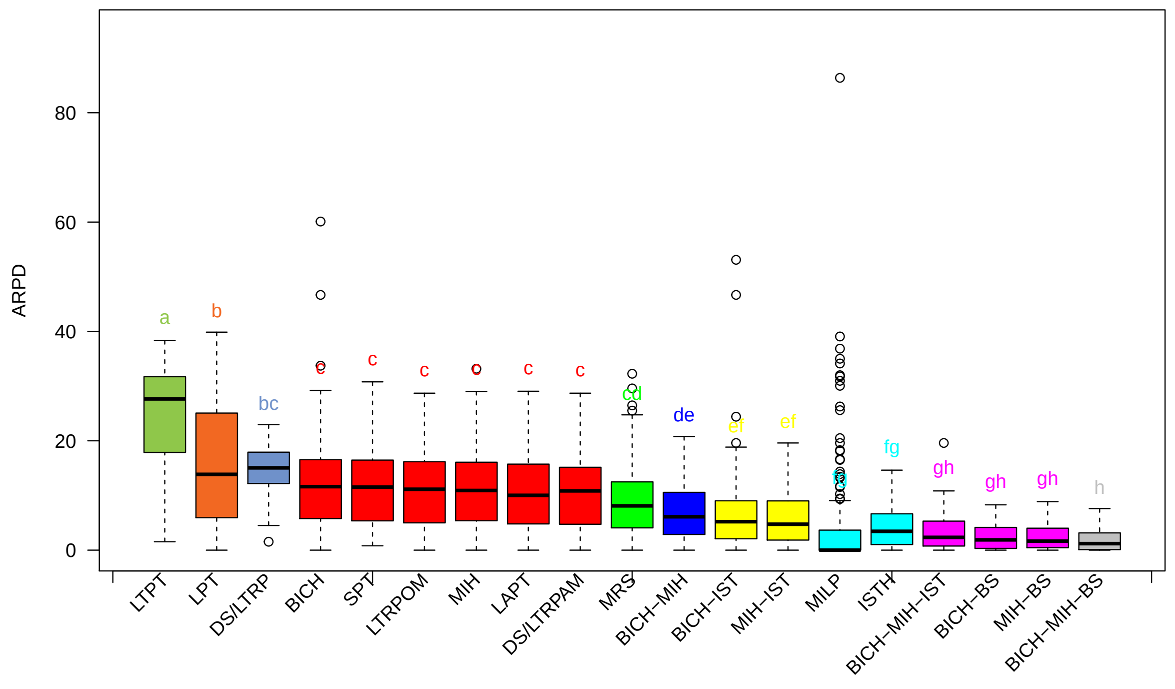
Regarding computational times, Table 8 shows the average computational times of constructive heuristics in each set of instances.

Figure 22 – Benchmark of constructive heuristics for each set of instances proposed by Brucker *et al.* (1997).



Source: Authors

Figure 23 – Boxplot and Tukey HSD groups at the 95% confidence level for the constructive heuristics in all sets of instances.



Source: Authors

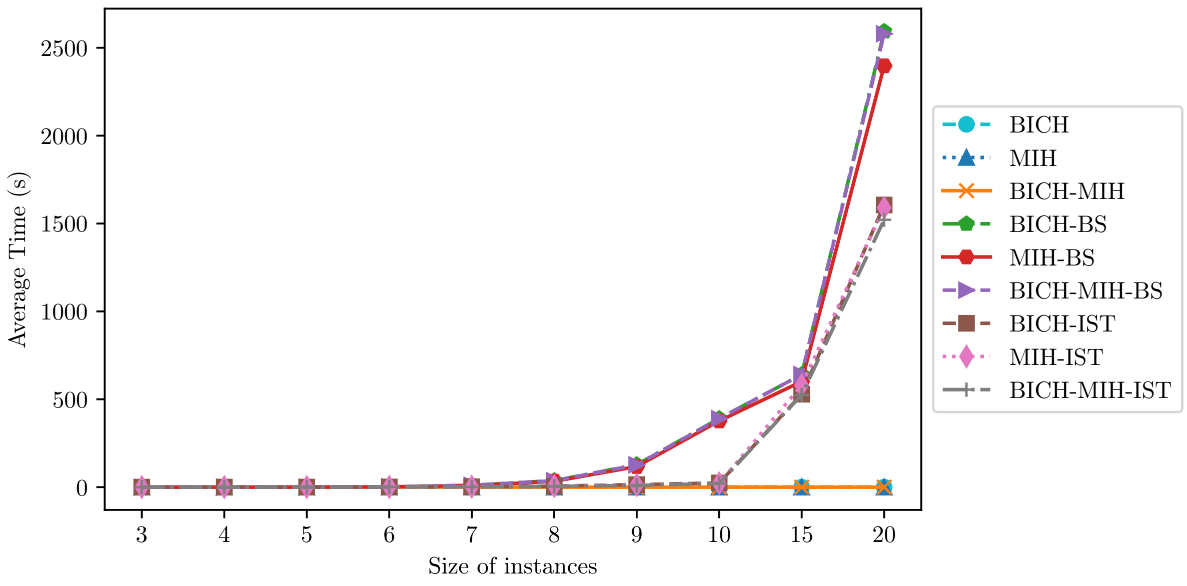
Table 8 – Computational times of constructive heuristics for each set of instances.

Benchmark	LPT	SPT	LAPT	LTPT	LTRPOM	DS/LTRP	DS/LTRPAM	MRS	ISTH	BICH	MIH	BICH-MIH	BICH-BS	MIH-BS	BICH-MIH-BS	BICH-IST	MIH-IST	BICH-MIH-IST
Guéret and Prins																		
GP-03	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	0.07	0.04	0.04	0.02	0.02	0.02
GP-04	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	0.64	0.64	0.58	0.07	0.06	0.05
GP-05	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	7.85	7.81	6.95	0.16	0.15	0.14
GP-06	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	52.39	52.75	46.96	0.45	0.44	0.44
GP-07	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	318.57	314.98	290.10	1.27	1.27	1.26
GP-08	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	598.94	597.44	528.89	3.67	3.44	3.25
GP-09	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	868.45	871.00	829.78	7.88	7.71	7.12
GP-10	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	1284.96	1324.42	1127.94	17.33	17.39	16.28
Taillard																		
tai_4x4	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	0.62	0.61	0.57	0.04	0.04	0.04
tai_5x5	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	7.85	7.81	7.18	0.16	0.16	0.15
tai_7x7	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	294.73	300.38	273.18	1.35	1.27	1.25
tai_10x10	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	1111.83	1132.63	1002.33	18.24	17.97	16.59
tai_15x15	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	1504.02	1511.32	1422.52	388.13	380.79	349.85
tai_20x20	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	1933.77	1951.65	1852.64	1234.93	1222.78	1170.42
Brucker																		
j3	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	0.05	0.04	0.04	0.01	0.01	0.01
j4	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	0.67	0.67	0.60	0.04	0.04	0.04
j5	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	8.35	8.21	7.71	0.14	0.14	0.14
j6	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	51.52	54.03	46.94	0.44	0.44	0.44
j7	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	297.71	302.14	273.28	1.26	1.26	1.25
j8	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	545.49	560.42	483.02	3.27	3.27	3.25
Min	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	0.05	0.04	0.04	0.01	0.01	0.01
Average	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	444.42	449.95	410.06	83.94	82.93	78.60
Max	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	<1.00	1933.77	1951.65	1852.64	1234.93	1222.78	1170.42

Source: Authors.

Figure 24 illustrates the dependence between solution time and problem size for the constructive heuristics. This figure presents the average computational for each evaluated solution procedure for each size of instance. In the smaller test instances we have 3 machines and 3 jobs (9 operations), and in the larger test instances we have 20 machines and 20 jobs (400 operations).

Figure 24 – Dependence between solution time and problem size for the constructive heuristics.



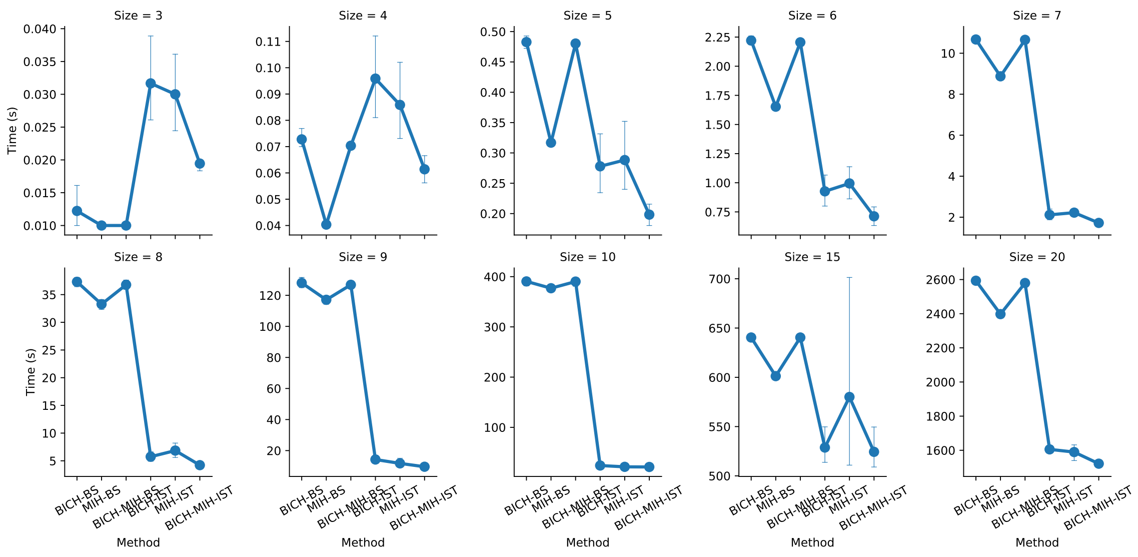
Source: Authors

We can observe that the computation time increases exponentially when the number of machines and jobs is greater than 10. This trend is even more evident for the solution procedures based on IST and BS. For instance sizes with 15 or more machines and jobs, the IST-based algorithms present smaller computation times than the BS-based algorithms. Thus, the IST-based algorithms still are competitive with computation times less than 1700 seconds. The increase of the problem size does not imply a substantial augment in the computation times for the IST and BS algorithms.

Figure 25 illustrates the average computation times with a 95% confidence interval for each constructive heuristic. For the test instances with less than 5 machines and jobs, the BS-based algorithms have lower computation times than the IST-based algorithms. Concerning larger instance sizes, IST-based algorithms present lower computation times than the BS-based algorithms. We can observe that this trend becomes more evident with the increase of instance sizes. In summary, we can conclude that the constructive heuristics based on IST and BS algorithms can treat problems presenting less than 15 machines and jobs with adequate computation times (approximately 10 minutes).

The Pareto Chart of average computational times and ARPD of our proposed

Figure 25 – Average computation times for the constructive heuristics.



Source: Authors

methods is presented in Figure 26. As it can be seen, the proposed IST approach presents a better combination of solution quality and computational efficiency. The constructive heuristics proposed by Abreu *et al.* (2020) present the best computation times, while the hybridization of BICH-MIH with BS proposed in this paper gives the best ARPD results.

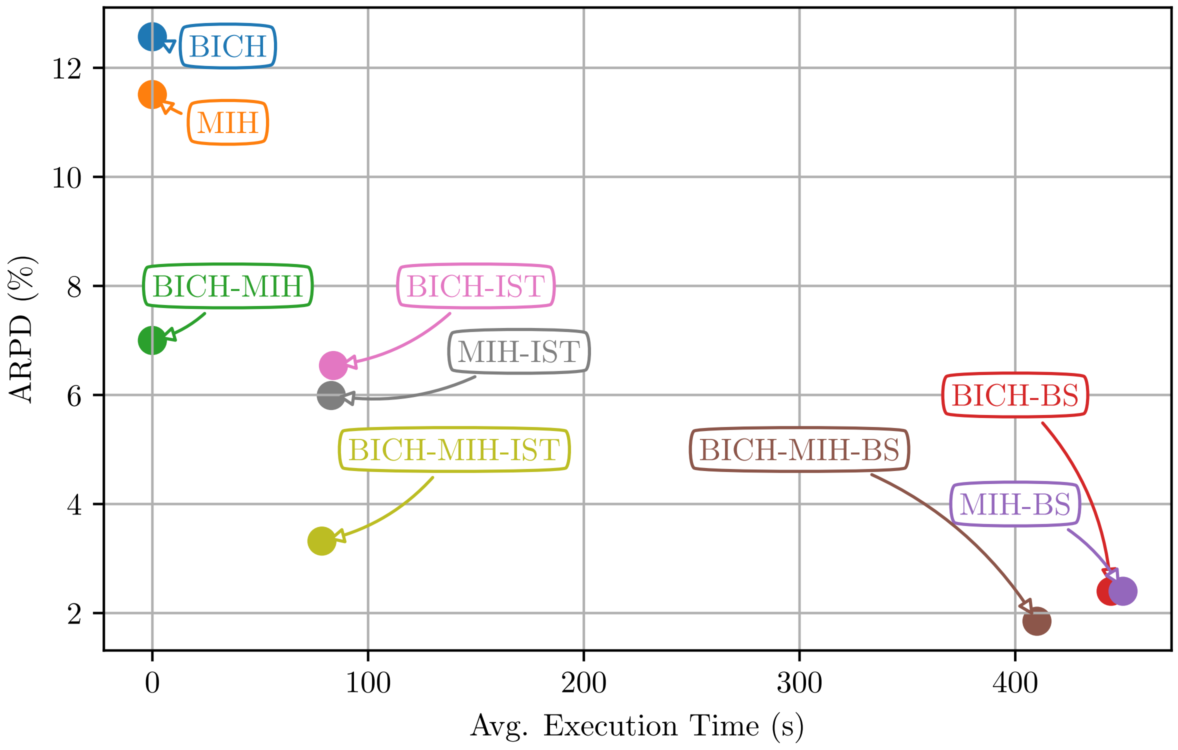
The hybridization of the heuristics with BS and IST procedure outperforms the BICH and MIH algorithms, being the differences statistically significant as they are in different groups in Figure 23. Therefore, the BICH-MIH-BS turns out to be the best constructive heuristic for the OSSP with a good trade-off between solution quality and computational times.

5.4.2 Computational results for local search heuristics

In this section, we evaluate the performance of the following local search/metaheuristic algorithms (in addition, the MILP model is considered for comparison purposes):

- Insertion and Reinsertion Heuristic 1-4 (IRH1 to IRH4): local search algorithms proposed by Naderi *et al.* (2010).
- Bounded Insertion Constructive Heuristic followed by local search (BICH-LS).
- Minimal Insertion Heuristic followed by local search (MIH-LS).
- Combined algorithm followed by local search (BICH-MIH-LS).
- Bounded Insertion Constructive Heuristic with Beam Search procedure followed by local search (BICH-BS-LS).

Figure 26 – Pareto Chart for Average computational times and ARPD of proposed constructive heuristics.



Source: Authors

- Minimal Idleness Heuristic with Beam Search procedure followed by local search (MIH-BS-LS).
- Combined algorithm with Beam Search procedure followed by local search (BICH-MIH-BS-LS).
- Bounded Insertion Constructive Heuristic with Cheap Insertion procedure followed by local search (BICH-IST-LS).
- Minimal Idleness Heuristic with Cheap Insertion procedure followed by local search (MIH-IST-LS).
- Combined algorithm with Cheap Insertion procedure followed by local search (BICH-MIH-IST-LS).
- The genetic algorithm EGA proposed by Hosseinabadi *et al.* (2018).

A summary of the computational results is presented in Table 9. The results for the test instances of Guéret and Prins, Taillard and Brucker are presented in Figures 27, 28 and 29 respectively.

Table 9 – Results of constructive heuristics with local search, meta heuristics and mixed integer programming for each set of instances.

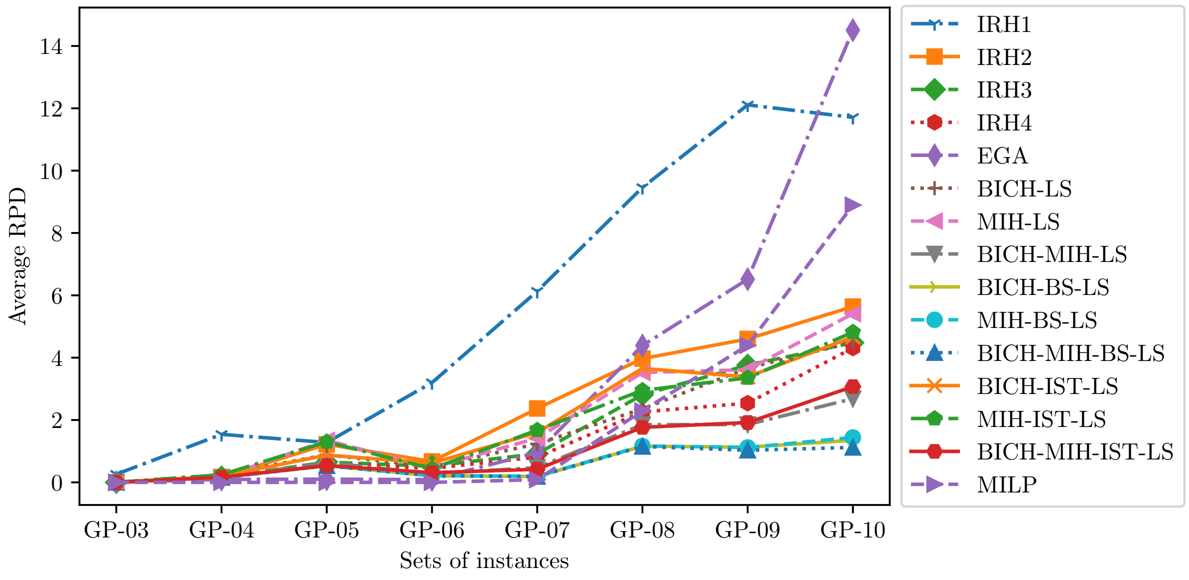
Benchmark	MILP	IRH1	IRH2	IRH 3	IRH 4	EGA	BICH-LS	MIH-LS	BICH-MIH-LS	BICH-BS-LS	MIH-BS-LS	BICH-MIH-BS-LS	BICH-IST-LS	MIH-IST-LS	BICH-MIH-IST-LS
Guéret and Prins															
GP-03	**0.00**	0.26	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**
GP-04	**0.00**	1.54	0.17	0.17	0.17	0.10	0.24	0.16	0.16	0.16	0.16	0.16	0.20	0.24	0.17
GP-05	**0.00**	1.28	1.22	0.65	0.60	0.10	0.89	1.35	0.63	0.53	0.53	0.53	0.87	1.30	0.54
GP-06	**0.00**	3.20	0.67	0.52	0.45	0.09	0.50	0.47	0.28	0.21	0.21	0.21	0.62	0.44	0.31
GP-07	**0.09**	6.13	2.37	0.92	0.81	0.96	1.24	1.42	0.47	0.19	0.20	0.19	1.60	1.67	0.42
GP-08	2.29	9.46	3.97	2.81	2.26	4.39	2.36	3.52	1.85	1.16	1.17	**1.15**	3.66	2.95	1.76
GP-09	4.40	12.11	4.60	3.76	2.54	6.52	3.67	3.61	1.86	1.13	1.12	**1.03**	3.38	3.35	1.92
GP-10	8.90	11.71	5.64	4.48	4.31	14.51	4.50	5.41	2.68	1.35	1.44	**1.12**	4.66	4.82	3.07
Taillard															
tai_4x4	**0.00**	6.68	4.65	3.43	3.67	0.70	6.08	2.84	2.12	2.09	2.09	2.09	3.08	2.84	2.52
tai_5x5	**0.00**	9.11	8.52	5.42	4.24	1.30	3.89	4.24	2.38	1.94	1.94	1.94	3.14	4.43	2.38
tai_7x7	1.28	8.07	6.43	3.02	4.12	3.98	3.04	3.52	1.63	0.43	0.47	**0.27**	3.05	2.50	1.56
tai_10x10	3.44	7.89	4.92	1.67	1.96	5.00	1.56	1.93	0.95	0.25	0.16	**0.02**	1.08	1.49	0.41
tai_15x15	17.10	-	-	-	-	7.59	0.67	0.42	0.21	0.25	0.28	**0.09**	0.26	0.28	**0.09**
tai_20x20	38.25	6.06	1.40	0.39	0.80	13.57	0.35	0.32	0.10	0.10	0.08	**0.02**	0.10	0.08	**0.02**
Brucker															
j3	**0.00**	0.80	3.56	3.56	3.56	0.01	15.92	4.73	2.62	2.50	1.89	0.88	8.95	4.73	2.62
j4	**0.00**	3.92	5.47	3.06	2.65	**0.00**	3.09	2.54	2.01	1.82	1.82	1.82	2.94	1.99	1.82
j5	**0.00**	9.00	3.37	2.17	4.44	1.14	5.10	4.03	2.09	1.27	1.27	1.27	4.76	3.71	2.36
j6	**0.00**	10.00	6.05	3.40	3.21	4.56	8.99	5.38	2.74	3.87	1.41	1.39	8.88	6.22	2.04
j7	2.92	12.40	7.78	3.83	4.58	6.87	7.19	5.80	3.55	3.71	2.09	**1.95**	6.01	5.39	3.02
j8	7.09	12.27	7.16	5.31	5.10	8.97	6.11	5.85	4.19	2.63	2.04	**1.83**	5.11	5.02	3.37
Min	**0.00**	0.26	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**
Average	4.29	6.94	4.10	2.56	2.60	4.02	3.77	2.88	1.63	1.28	1.02	**0.90**	3.12	2.67	1.52
Max	38.25	12.40	8.52	5.42	5.10	14.51	15.92	5.85	4.19	3.87	**2.09**	**2.09**	8.95	6.22	3.37

Source: Authors.

In order to validate the results, as in the previous experiments, an ANOVA is applied in order to verifiy if the observed differences in the results of the local search algorithms are statistically significant. The p-value is very close to zero. We can see in Figure 30 the ARPD boxplot for all constructive heuristics with local search tested with HSD Tukey group ($\alpha = 0.05$) of the similar mean result. We can see that there are statistically significant differences between the ARPD values among the local search algorithms tested. The combined approach with IST and the constructive heuristics with BS gets the best results with medians very close to zero.

According to the results obtained for the eighty Guéret and Prins test problems, the following comments can be highlighted: The MILP returns the best solutions for small and medium sizes classes of instances and BICH-MIH-BS-LS returns the best solutions for large instances sizes with 8, 9 and 10 problems size. The IR1 method is the worst algorithm for the analyzed instances. The methods with beam search procedure outperforms BICH-LS, MIH-LS and BICH-MIH-LS, showing that the new approach to construct solutions gives better results. The BS algorithms give the best results with ARPD less than 1%. Considering only the constructive heuristics with local search, the proposed BICH-MIH-BS-LS algorithm presented the best average results for all the eight classes of instances.

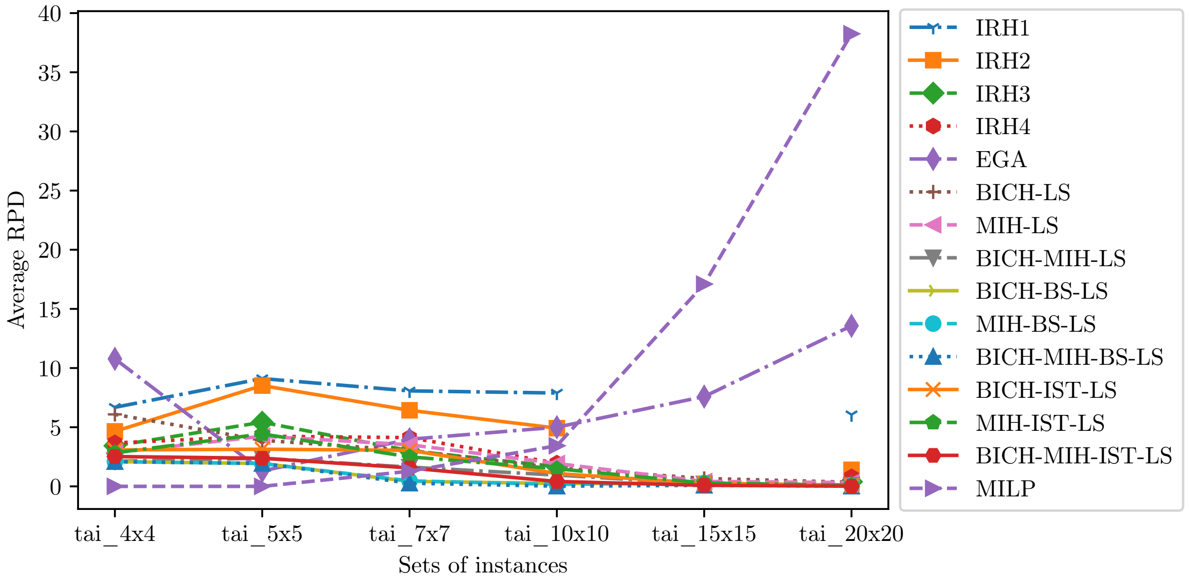
Figure 27 – Benchmark of constructive heuristics with local search for each set of instances proposed by Guéret and Prins (1999).



Source: Authors

With respect to the results for the sixty Taillard test problems, the following comments can be made. The MILP model returns the best average results only for the 4 and 5 problems sizes. In general, the IR1, IR2 and EGA are the worst algorithms for this set of problems. For large problems size, the MILP model returns the worst results. The behavior of all other constructive heuristics with local search is similar to the results for the Guéret and Prins test problems. The combined BICH-MIH-BS-LS presented the best results for the largest instances (tai_7x7, tai_10x10, tai_15x15 and tai_20x20).

Figure 28 – Benchmark of constructive heuristics with local search for each set of instances proposed by Taillard (1993).



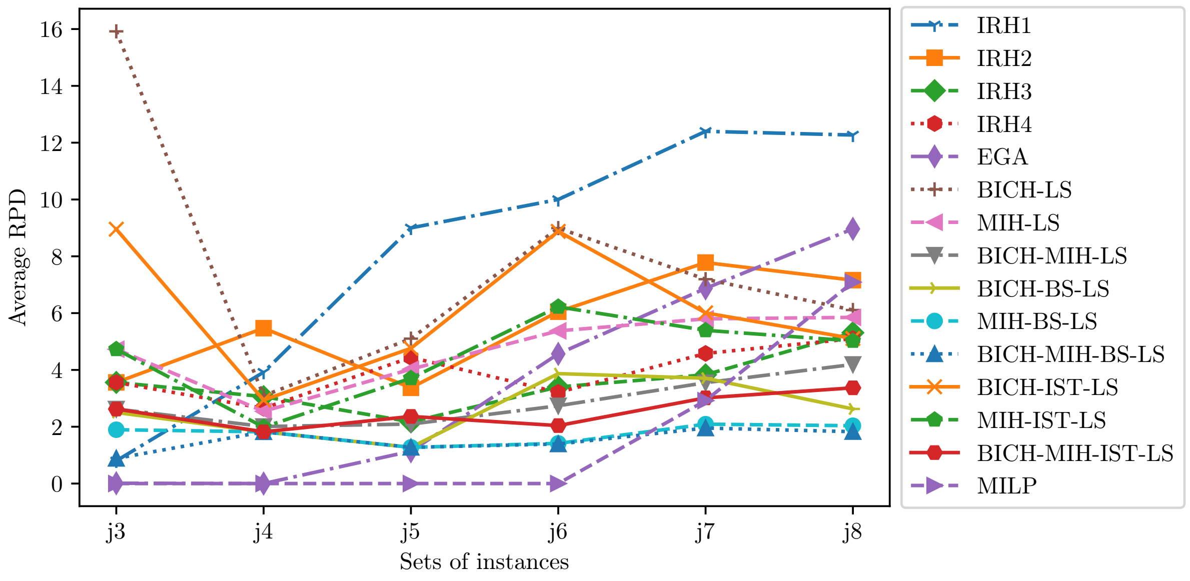
Source: Authors

For the Guéret and Prins instances, as well as for the Taillard instances, the proposed algorithms with beam search and cheapest insertion procedure present better results than IRx and EGA. In general, the MILP method obtains the better results for the small-sized instances and the BICH-MIH-BS-LS present the best results for the large-sized instances.

Finally, for the Brucker instances, the MILP method outperforms the other evaluated methods, with the exception of the j7 and j8 sets of instances, in which the BICH-MIH-BS-LS algorithm returns the best results.

On average, the MILP method yields poor results, because the model returned low-quality results for the large-sized test problems in the allotted CPU time. The proposed BICH-MIH-BS-LS algorithm outperforms all the other evaluated methods, showing that the combined approach with a weighted aggregation function is more efficient. Overall, the proposed approach BICH-MIH-BS-LS presented the lower ARPD for all the analyzed constructive heuristics as well as lower than the MILP model. Furthermore, the differences among BICH-MIH-BS-LS, EGA, MILP and methods proposed by Naderi *et al.* (2010) and Abreu *et al.* (2020) are significant because they are in different groups with different letters. The methods with beam search procedure and BICH-MIH-IST-LS present the lowest mean values for ARPD.

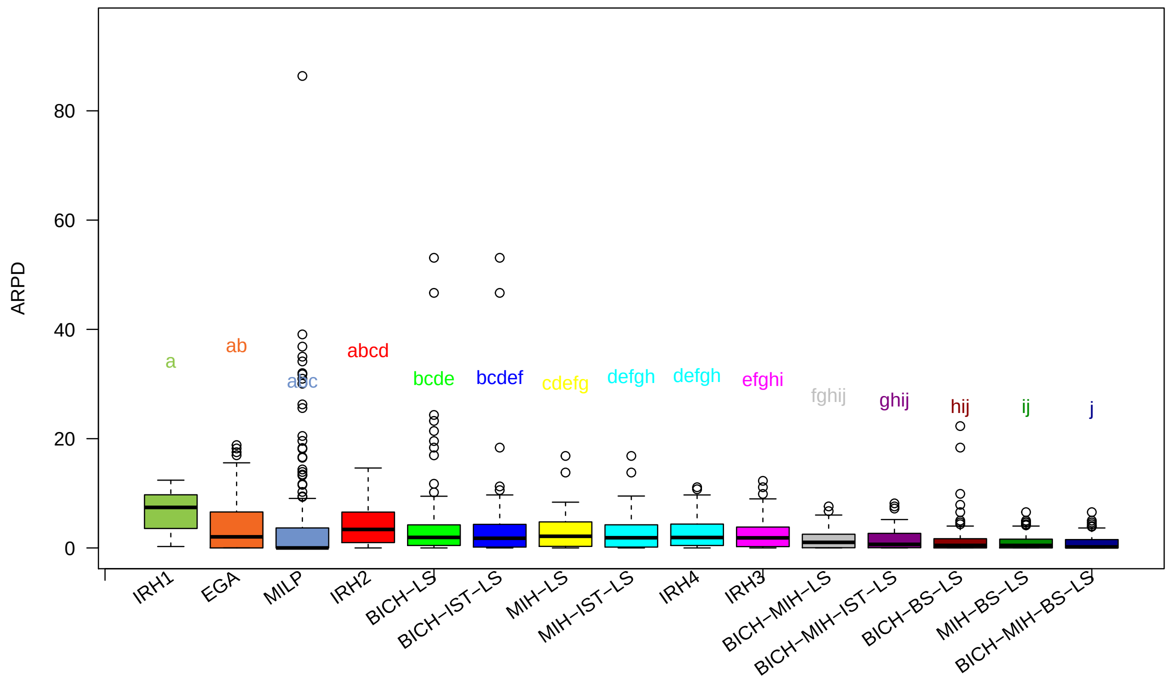
Figure 29 – Benchmark of constructive heuristics with local search for each set of instances proposed by Brucker *et al.* (1997).



Source: Authors

Regarding computational times, Table 10 shows the average computational times of constructive heuristics and metaheuristics in each set of instances.

Figure 30 – Boxplot and Tukey HSD groups at the 95% confidence level for the constructive heuristics with local search in all sets of instances.



Source: Authors

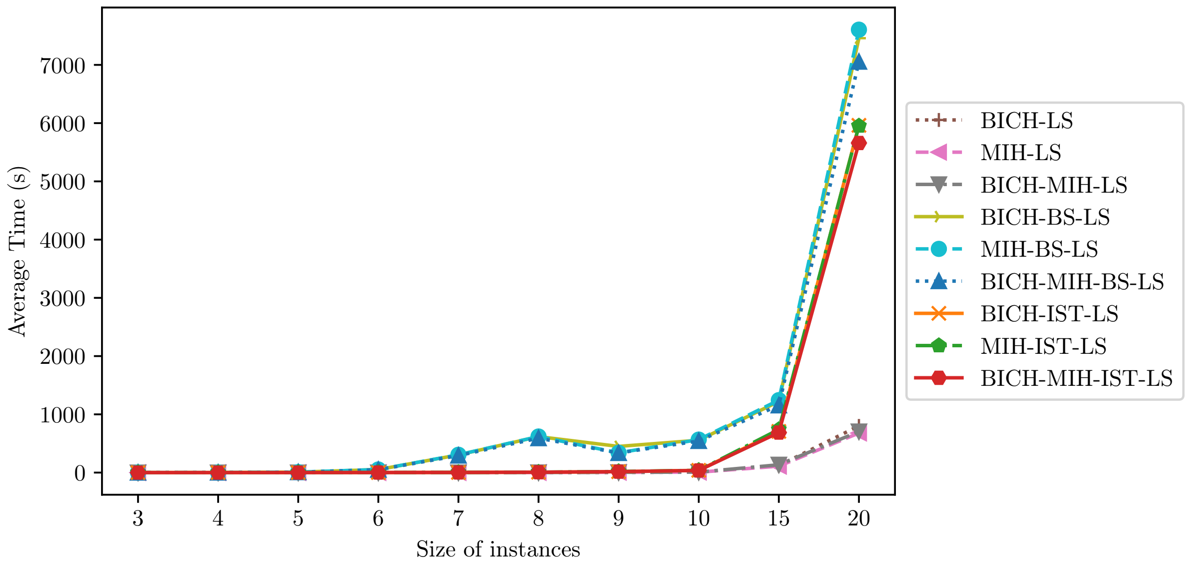
Table 10 – Computational times of constructive heuristics with local search and meta heuristics for each set of instances.

Benchmark	IRH1	IRH2	IRH 3	IRH 4	EGA	BICH-LS	MIH-LS	BICH-MIH-LS	BICH-BS-LS	MIH-BS-LS	BICH-MIH-BS-LS	BICH-IST-LS	MIH-IST-LS	BICH-MIH-IST-LS
Guéret and Prins														
GP-03	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.13	0.08	0.07	0.02	0.02	1.64
GP-04	0.00	0.00	0.01	0.01	36.76	0.01	0.01	0.01	1.21	1.22	1.11	0.09	0.09	1.47
GP-05	0.00	0.01	0.03	0.05	108.36	0.04	0.03	0.04	15.06	14.87	13.07	0.55	0.49	1.27
GP-06	0.00	0.02	0.11	0.15	134.51	0.15	0.12	0.16	99.52	99.49	89.42	1.06	1.14	1.73
GP-07	0.01	0.06	0.37	0.47	196.41	0.43	0.38	0.46	607.42	599.72	556.41	3.15	3.16	3.86
GP-08	0.02	0.15	1.14	1.20	300.01	1.17	1.02	1.24	1140.86	1139.38	1005.18	6.20	6.37	6.85
GP-09	0.03	0.35	2.77	2.72	300.01	2.41	2.92	2.72	1655.42	1669.84	1560.10	15.75	18.35	17.42
GP-10	0.05	0.66	4.19	5.43	300.01	4.61	4.82	5.54	2439.32	2507.13	2139.55	39.68	39.62	40.19
Taillard														
tai_4x4	0.00	0.00	0.01	0.01	192.29	0.01	0.01	0.01	1.18	1.16	1.08	0.07	0.06	1.40
tai_5x5	0.00	0.01	0.04	0.05	254.16	0.04	0.04	0.04	14.99	14.73	13.81	0.26	0.23	1.51
tai_7x7	0.01	0.06	0.39	0.49	300.01	0.32	0.38	0.37	563.31	572.50	517.97	2.16	2.56	3.09
tai_10x10	0.05	0.68	4.47	5.54	300.01	4.59	3.90	5.26	2140.05	2146.40	1885.65	37.98	37.17	38.15
tai_15x15	-	-	-	-	300.03	73.27	66.62	81.98	2849.63	2880.84	2702.72	706.82	739.87	702.76
tai_20x20	1.18	79.46	517.91	707.79	300.06	449.55	378.03	395.84	3660.44	3755.26	3506.56	3731.02	3619.92	3616.29
Brucker														
j3	0.00	0.00	0.00	0.00	37.50	0.00	0.00	0.00	0.10	0.08	0.07	0.02	0.02	1.39
j4	0.00	0.00	0.01	0.01	67.73	0.01	0.01	0.01	1.27	1.26	1.15	0.09	0.08	1.54
j5	0.00	0.01	0.04	0.05	222.09	0.04	0.03	0.04	15.85	15.73	14.60	0.35	0.34	1.58
j6	0.01	0.02	0.13	0.16	300.00	0.16	0.13	0.18	97.82	101.85	89.28	1.31	0.96	1.97
j7	0.01	0.06	0.37	0.46	300.01	0.37	0.33	0.47	569.31	571.65	516.13	2.85	2.95	3.66
j8	0.02	0.15	0.94	1.10	300.01	0.85	0.82	1.06	1031.35	1060.00	904.93	7.31	8.19	8.52
Min	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.10	0.08	0.07	0.02	0.02	1.27
Average	0.08	4.63	30.46	41.08	212.50	26.90	22.98	24.77	845.21	857.66	775.94	227.84	224.08	222.81
Max	1.18	79.46	517.91	707.79	300.06	449.55	378.03	395.84	3660.44	3755.26	3506.56	3731.02	3619.92	3616.29

Source: Authors.

With respect to computational times for the constructive heuristics with local search, Figure 31 illustrates the dependence between solution time and problem size for the constructive heuristics with local search. We can observe that the BS-based algorithms have started to increase their computation times from the instances with 8 or more machines and jobs. For problem sizes with 15 or more machines and jobs, we can observe that all the constructive heuristics with local search increased their computation times.

Figure 31 – Dependence between solution time and problem size for the constructive heuristics with local search.



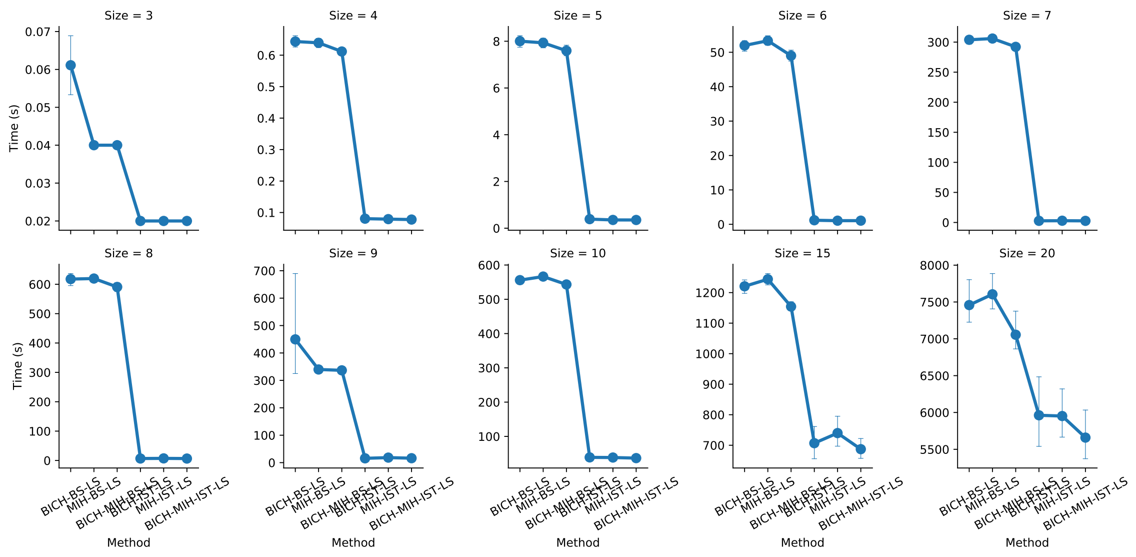
Source: Authors

Figure 32 illustrates the average computation times with a 95% confidence interval for each constructive heuristic with local search. In contrast with the constructive heuristics without local search, in this situation, the IST-based heuristics present computation times substantially smaller than the BS-based heuristics. The two classes of algorithms can deal with problems with less than 15 jobs and machines (around 2000 s).

As a conclusion of the computational time analysis, IST- and BS-based constructive heuristics can handle 20x20 instances in less than one hour while their local search counterpart would require more than double of the time. Since the computation times around this instance size grow rapidly, we believe that 20x20 (which represents a total of 400 operations in the shop) represents the limit in the problem size that, for many decision scenarios, can be realistically addressed in a standard computer.

The Pareto Chart of average computational times and ARPD is presented in Figure 33. On the basis of the above, the proposed IST approach presents a better combination of solution quality and computational efficiency. The constructive heuristics proposed by Abreu *et al.* (2020) with local search and IRx algorithms presents best computation times and BS approach presents the best ARPD results.

Figure 32 – Average computation times for the constructive heuristics with local search.



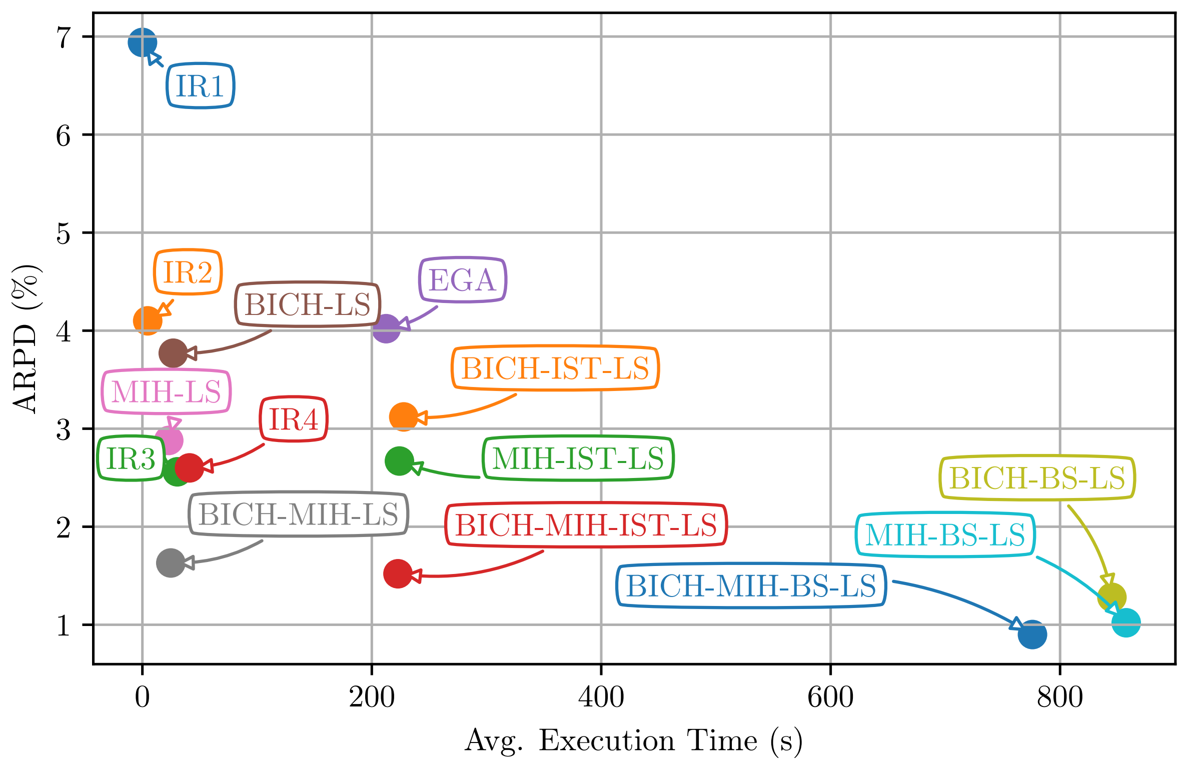
Source: Authors

The constructive heuristics with BS and IST procedure outperforms the classic approach of BICH and MIH algorithms proposed by Abreu *et al.* (2020), IRx algorithms proposed by Naderi *et al.* (2010) and EGA proposed by Hosseinabadi *et al.* (2018). The BICH and MIH with BS procedure gives the best results than BICH and MIH with IST procedure, but the algorithms with IST procedure gets good computational times, becoming a good choice for industrial applications with operational level of scheduling problems. The BICH-MIH-BS-LS becoming the best constructive heuristic with local search for OSSP.

As a summary of the performance of the BS-based algorithms, note that this family of algorithms obtains very low ARPD values at the costs of requiring more CPU time. As illustrated in Figure 26, the inclusion of beam search procedures resulted in a substantial improvement in the quality of the solutions found when compared to the algorithms considering cheap insertion procedures, even if it is to note that this increases the computation times. In Figure 26, we can also observe that the BICH-MIH-BS dominated the MIH-BS and BICH-BS algorithms.

Concerning the local search versions of the proposed algorithms, it can be seen in Figure 33 that, when compared to IR3 and IR4 – the best versions of IRx algorithms – the BICH-MIH-LS presented lower ARPD values with a similar average computational time. With respect to the ARPD values, the BICH-MIH-BS-LS algorithm presented the best results, dominating the MIH-BS-LS and BICH-BS-LS algorithms.

Figure 33 – Pareto Chart for Average computational times and ARPD of proposed constructive heuristics with local search.



Source: Authors

5.5 Concluding remarks

In this paper, we focus on the classical variant of the OSSP. The objective function is to minimize the total time to complete the schedule (makespan). We develop new beam search and cheapest insertion procedures hybridized with constructive heuristics adapted from the problem with setup considerations. Finally, an efficient local search algorithm (LS) that leads to excellent results within an admissible computational effort is proposed.

A number of computational experiments were carried out in order to evaluate the performance of the proposed algorithms. The results of the proposed approaches are presented considering the literature benchmark instances proposed by Guéret and Prins (1999), Taillard (1993) and Brucker *et al.* (1997). We used the relative percentage deviation statistics as performance measure. Taking into consideration the above mentioned literature benchmark instances, the proposed constructive algorithm BICH-MIH-BS outperforms the existing constructive heuristics, the BICH-MIH-BS-LS algorithm outperforms the four local search algorithms proposed by Naderi *et al.* (2010) and the genetic algorithm proposed by Hosseinabadi *et al.* (2018).

As extensions of this work, we suggest the consideration of explicit travel times and resource utilization in the OSSP to address more realistic environments. In addition, future studies could also investigate the behavior of the proposed approaches considering different objective functions, such as total tardiness minimization with due dates of jobs or total completion time minimization.

6 A NEW HYBRIDIZATION OF ADAPTIVE LARGE NEIGHBORHOOD SEARCH WITH CONSTRAINT PROGRAMMING FOR OPEN SHOP SCHEDULING WITH SEQUENCE-DEPENDENT SETUP TIMES

Production scheduling is an area of production planning and control responsible for determining how tasks will be allocated to existing resources over scheduling horizon (MACCARTHY; LIU, 1993). This area is constituted by a set of optimization problems having a series of real applications.

Production scheduling problems are widely studied as optimization problems due to various industrial applications. An open shop scheduling problem (OSSP) consists of scheduling a set of jobs in a set of machines, where each operation is associated with a processing time and there is no predefined sequence of operations to execute. The problem is of practical and theoretical importance and less attention is paid to traditional production scheduling problems (ANAND; PANNEERSELVAM, 2016; ADAK; AKAN; BULKAN, 2020; AHMADIAN *et al.*, 2021a).

As the problem does not have a predefined sequence of operations, the number of viable solutions is significantly higher than other production scheduling problems, such as flow shop and job shop (AHMADIAN *et al.*, 2021a). OSSP has many industrial applications, such as plastic injection, chemical processes, oil industries, food production, joint scheduling with low-carbon emissions and pharmaceuticals. In the service sector, this problem can be modeled for scheduling medical services, vehicle maintenance, museum visits, sharing economy, healthcare diagnostics, and telecommunications (GONZALEZ; SAHNI, 1976; LIN; LEE; PAN, 2008; VINCENT; LIN; CHOU, 2010; NADERI; ZANDIEH, 2014; CANKAYA; WARI; TOKGOZ, 2019; FU *et al.*, 2021; ABDELMAGUID, 2021; AGHIGHI *et al.*, 2021; ABREU *et al.*, 2021; KURDI, 2022).

Among the industrial applications of the open shop, we can highlight the study of the open shop problem with reverse flows in an assembly of computer boards and chips industry with assembly and dismounting operations of electronic security alarm boards presented by Aghighi *et al.* (2021) and the study of scheduling and handling in the transportation of chemical tanks in ports presented by Cankaya, Wari and Tokgoz (2019).

Among the applications in the service sector, we can highlight the study of the museums visitor scheduling problem where groups of visitors are jobs that can visit exhibitions in any sequence presented by Vincent, Lin and Chou (2010). In addition, Abreu *et al.* (2021) presented the study of scheduling vehicle maintenance operations in workstations where there can be repetitions of operations in the same workstation.

In traditional production scheduling research, setup times are implicit, that is, they are considered part of the processing time. However, this consideration may imply an increase in the total schedule duration when setup times are sequence-dependent, as the planning obtained can result in a high occurrence of machine setup activities (ALLAHVERDI, 2015).

Therefore, when sequence-dependent setup times are considered explicit production scheduling problems can be solved more realistically, and schedules that reduce these machine setup times can be developed (ALLAHVERDI *et al.*, 2008).

Despite the theoretical and practical importance of the OSSP with explicit setup times, contributions in the literature are limited. These limitations have been highlighted in some recent literature reviews on production scheduling problems with setup costs (ALLAHVERDI; GUPTA; ALDOWAISAN, 1999; ALLAHVERDI *et al.*, 2008; ALLAHVERDI, 2015) and the OSSP (ANAND; PANNEERSELVAM, 2016; ADAK; AKAN; BULKAN, 2020; AHMADIAN *et al.*, 2021a).

Therefore, this research aims to study open shop scheduling with non-anticipatory sequence-dependent setup time (OSSPST) considering its basic structure with explicit machine setup times, that is, the duration of the setup time of a job on a machine depends on the previous job that was processed on the machine. The setup time is non-anticipatory as one has to wait for the current task to finish processing on the previous machine to start preparing it on the next machine. As a practical example, there may be adjustments on the next machine, in which the job needs to be present (FRAMINAN; LEISTEN; GARCÍA, 2014, chapter 4).

OSSPST aims to minimize the scheduling duration (makespan). For its resolution, we applied new algorithms based on the hybridization of exact and approximate strategies, comparing the performance of the proposed methods with other competitive methods in the literature, leading to improvements in instance sets already reported for the problem.

In summary, this paper aims to present a hybridization of adaptive large neighborhood search with constraint programming (ALNS-CP) for the open shop with setup times and makespan minimization as the objective function. The main theoretical contributions are: The proposal of a new constraint programming model using knowledge of the problem, such as the type of setups, to reduce the number of variables and constraints. We propose, for ALNS-CP, an adaptive destruction mechanism prioritizing the destruction's operators that return the best makespan results along iterations of the method. In addition, ALNS-CP has a solution reconstruction mechanism that uses the new constraint programming model, receiving a partial solution of the problem and returning an improved feasible solution.

The main experimental findings are: The new proposed CP model gets competitive results in quality and computational times when compared with all other exact methods tested. We propose testing several emergent metaheuristics for the open shop with setups.

These metaheuristics have never been fully applied together in the literature. ALNS-CP has an average relative percentage deviation smaller than 22% in tested instances, obtained good results in large-sized instances, and ALNS-CP has competitive computational times compared to other benchmarking methods.

The article is structured as follows: Section 6.1 reviews the OSSPST literature, Section 6.2 illustrates the formal definition of the problem and the exact resolution models. Section 6.3 describes the proposed resolution method. Section 6.4 performs an extensive analysis of the results and statistical tests. Finally, Section 6.5 comments on the main conclusions of the research and proposes future studies on the problem.

6.1 Literature review

As open shop scheduling is NP-Hard (GAREY; JOHNSON, 2012), the main solving techniques are the approximate algorithms, although there are exact applications for OSSP, considering trivial cases (such as unit setup times). Below, we discuss the main contributions related to OSSP with explicit setup times.

Allahverdi, Gupta and Aldowaisan (1999) carried out a literature review on production scheduling problems with explicit setup time constraints. For the case of open shop scheduling, only one article was proposed at the time. For the second survey carried out by the author, eight more articles were mentioned (ALLAHVERDI *et al.*, 2008). In the last study carried out, four more articles that involved the problem were mentioned. This shows that the topic has not yet been fully explored in the literature (ALLAHVERDI, 2015).

The first article to explicitly consider open shop setup times was that of Strusevich (1993), who considered the case for open shop scheduling with two machines and times of: setup, processing and job removal. The authors developed an accurate O(n) polynomial time algorithm for makespan minimization.

Other articles aimed at developing exact strategies for solving special cases of the problem, considering the system with two or three machines or other trivial constraints (STRUSEVICH, 1999; BLAZEWICZ; KOVALYOV, 2002; BRUCKER *et al.*, 2004; PANWALKAR; KOULAMAS, 2014; BEVERN; PYATKIN, 2016; BABOU; REBAINE; BOUDHAR, 2021).

Exact solving strategies define good properties for problem knowledge. However, in certain cases, these methods are only suitable for special open shop cases, simplifying the main restrictions such as: small numbers of machines, constant or machine-independent processing times, and fixed or independent setups. Therefore, for the problem proposed in this article, arising mainly from complex real systems, approximate strategies need to be used, such as heuristics and metaheuristics.

To solve problems with real constraints, using many machines, jobs and constraints, approximate algorithms need to be used. Among them are priority rules, constructive solution strategies and nature-inspired optimization algorithms.

Lee and Malone (2000) propose the first meta-heuristic for the problem, considering batch setups. Simulated Annealing (SA) is proposed to compare the results with classic priority rules, such as: FIFO, SPT and EDD. The instances used came from a case study of an industry. The proposed algorithm performed better than the priority rules.

Roshanaei, Esfehani and Zandieh (2010) addressed the OSSP with sequence-dependent setup constraints, using SA with multiple neighborhoods and adaptive constructive heuristics LAPT and LRPT for the setup problem. To test the methods, the authors adapted classical benchmarking instances for the problem. The proposed SA obtained a better performance when compared to classic metaheuristics from the literature, also adapted for the problem.

Naderi *et al.* (2011a) developed an electromagnetism-like meta-heuristic (EH) for open shop scheduling with sequence-dependent setup times to minimize the total completion time (TCT). This objective has not been addressed in previous studies with approximate methods. EH performs well for OSSP problems and is adapted for cases with setups (NADERI; NAJAFI; YAZDANI, 2012). The authors compared the proposed algorithm with a mixed-integer linear programming (MILP) model and obtained significant results, analyzed by statistical tools. A new solution decoding scheme based on non-delay programming was used, which seeks to anticipate the scheduling of operations respecting their relative positions found in the solution of the problem.

Vincent, Lin and Chou (2010) explored a real problem that could be modeled as an OSSP with setup times. The museum visitor routing problem, where each group of visitors (jobs) visits a set of exhibitions (machines) within a museum, the transport time from one exhibition to another, and the exhibition's duration can be modeled as setup and processing time, respectively. The authors propose SA, aiming to minimize the total duration of visits, such as the makespan minimization. The SA algorithm presented acceptable computational quality and times and was able to be used in practical applications.

Zhuang *et al.* (2019a) implemented a network-based constructive heuristic for solving the OSSP with sequence-dependent setup times and job delivery times between machines. The algorithm was compared with an MILP and classical metaheuristics such as the artificial bee colony (ABC) proposed by Zhuang *et al.* (2019b). The constructive heuristic obtained a better performance than the other methods for the makespan minimization.

Cankaya, Wari and Tokgoz (2019) proposed to solve a case study of chemical tanker scheduling in the port of Houston (USA) as an OSSP with setup times based on cleaning times and transport times between terminals. These times are dependent on the tank cleaning sequence. The authors proposed two exact methods for solving the problem, an MILP and a constraint programming model. The methods are compared with an empirical programming approach used by the rank, in this case the first in first out (FIFO) priority rule. The results show that the constraint programming presents a better performance than the empirical solution and the MILP, for normal operation conditions, which can be widely used in the real system.

Abreu *et al.* (2020) developed a hybrid genetic algorithm (GA) for the OSSP with sequence-dependent setup times to minimize the TCT. Three new constructive heuristics were also proposed based on characteristics of the problem domain, such as the lower bound. The authors optimize the GA parameters using the Taguchi experiment and combined the GA procedure with constructive heuristics. The GA was compared with the EH of Naderi *et al.* (2011a) and with the developed heuristics. Computational tests showed that the GA obtained significantly better results than all other methods.

Mejía and Yuraszeck (2020) studied the OSSP with sequence-dependent travel/setup times. The authors proposed a new decoding scheme and included it in a variable neighborhood search (VNS) metaheuristic. The authors tested the algorithm with new and classic instances from the literature, adapted for the problem, and compared it with another meta-heuristic and four constraint programming models with different search strategies. These methods were used to solve problems with makespan and TCT minimization. VNS got better results in both cases but did not have significantly better results than the constraint programming model. The new decoding scheme got a better result than that presented by Naderi *et al.* (2011a).

Finally, Behnamian, Dezfooli and Asgari (2021) presented a scatter search algorithm (SSA) for flexible open shop scheduling with independent setup times. The objective function is the minimization of the makespan and the total delay time (total tardiness), and is, therefore, a multiobjective problem. A new representation of a solution to the problem is proposed. The proposed algorithm presented better results when compared to classic multiobjective problem solving strategies, such as NSGA II.

Table 11 illustrates the main contributions of the literature to the OSSP considering setup times. The authors, year of publication, characteristics of the problem such as the type of setup, solution methods and main research contributions are illustrated.

Despite existing contributions (notably the EH algorithms by Naderi *et al.* (2011a), GA by Abreu *et al.* (2020), and VNS by Mejía and Yuraszeck (2020)), there is an opportunity to develop more efficient solution procedures for the problem based on the hybridization of exact and approximate methods and characteristics of the problem domain, such as the type of setup.

The present work aims to present a hybridization of adaptive large neighborhood

Table 11 – Summary of the main contributions from the OSSP literature considering setup times and metaheuristics approach.

Author	**Problem characteristics**	**Solution method**	**Contribution**
Lee and Malone (2000)	Batch setup times	SA and priority rules	Problem instances based in a industrial application.
Roshanaei, Esfehani and Zandieh (2010)	Dependent setup times	SA with multiple neighbohood and priority rules	SA outperforms all other tested methods.
Naderi *et al.* (2011a)	Dependent setup times	EH	A new decoded scheme for solutions.
Vincent, Lin and Chou (2010)	Dependent setup times and transportation times	SA	Real aplication of OSSP in museum visitor problem.
Zhuang *et al.* (2019a)	Dependent setup times and delivery times	Heuristic rule based on complex network	The heuristic rule outperforms all other tested methods.
Zhuang *et al.* (2019b)	Dependent setup times and delivery times	ABC	The ABC gets best results in large-scale instances.
Cankaya, Wari and Tokgoz (2019)	Dependent setup times	MILP and CP models	Real aplication of OSSP in tanks transports in ports.
Abreu *et al.* (2020)	Dependent setup times	GA and construtive heuristics	Heuristics was based on problem knowledge and domain.
Mejía and Yuraszeck (2020)	Dependent setup times	VNS and CP model	A new decoded scheme for solutions and VNS outperforms all other tested methods.
Behnamian, Dezfooli and Asgari (2021)	Independent setup times	SSA	Multiobjetive problem to minimize makespan and total tardness and a new solution representation.

Source: Authors.

search with constraint programming (ALNS-CP) for OSSPST and makespan minimization. To efficiently generate the initial solution, we used a constructive heuristic proposed by Abreu *et al.* (2020), which produces good solutions in competitive computational times. In the solution reconstruction phase, we uses a new CP model as a reconstruction and local search strategy. The new model uses features from the problem domain, such as non-anticipatory setups, to reduce the number of variables and constraints. The proposed CP model performed better than other CP models and the MILP model, already reported in the literature on the problem.

In order to determine the best configuration of the proposed ALNS-CP hybrid method, we used the IRACE package to determine the best execution parameters (LÓPEZ-IBÁNEZ *et al.*, 2016). Extensive computational experimentation was carried out using the well-known OSSP test instances proposed by the authors Guéret and Prins (1998), Taillard (1993) and Brucker *et al.* (1997), adapted to the problem by Abreu *et al.* (2020). The results show that the developed algorithm outperforms the existing procedures for the problem with competitive computational times.

6.2 Problem statement and exact models

OSSP was initially proposed by Gonzalez and Sahni (1976) and the objective was to scheduling a set of jobs in a set of machines, where each processing of a job in a machine (called an operation) had a processing time. The common objective for this problem was to minimize the total duration of the scheduling of all tasks, called makespan in the literature.

For the permutation flow shop scheduling problem, each job has the same route in each machine. On the other hand, in the job shop scheduling problem, each job has a different machine processing route. Unlike other classic production scheduling problems, in the OSSP there is no previous sequence for processing jobs on machines. The OSSP is like an job shop with no predefined routes for jobs. Therefore, the problem is constituted by a large number of viable solutions. In the classic problem, each job can only be processed by a single machine at a time, as well as jobs, once their processing starts, they cannot be interrupted (ABREU *et al.*, 2020).

Unlike other production scheduling problems, in the OSSP there is no previous sequence for processing jobs on machines, therefore, the problem is constituted by a large number of viable solutions. In the classic problem, each job can only be processed by a single machine at a time, as well as jobs, once their processing starts, they cannot be interrupted (GUÉRET; PRINS, 1998).

The variant of this article considers that between the processing time of one job and another on a machine, there is a setup time for the operation, called setup time, which depends on the previous job processed on the machine. The setup time is also non-anticipatory, as it considers that the job should have completed its processing in the previous machine to carry out the setup. It is also necessary that the job enters the next machine for the setup to be carried out. The objective function is makespan minimization. In the notation of Lawler *et al.* (1993) , the problem is defined as: $O_m|S_{sd}|C_{max}$.

As mentioned above, we use the makespan as a performance indicator in the article. An important indicator to be achieved in practical production scheduling situations is maximizing the utilization of production resources, such as machines, over the scheduling horizon. When we use makespan as the objective function of scheduling problems, we maximize the resource utilization by developing methods that reduce the makespan of the schedule (FRAMINAN; LEISTEN; GARCÍA, 2014), unlike minimizing total flow time or tardiness, that minimizes work-in-process inventory and job completion with tardiness, respectively.

In addition, by using makespan as an objective function, considering a static production environment such as the one proposed in the paper, it is possible to verify whether a given schedule can be performed in a given short-term time interval, such as a day or week (FRAMINAN; LEISTEN; GARCÍA, 2014). Furthermore, it is possible to quickly verify this information with the result of the makespan of the schedule created. Therefore, the objective function can generate agility for the decision-making process.

The data set for the problem consists of: a set of operations to be processed, where O_{ij} is the operation of job j on machine i, an $m \times n$ matrix of processing times p_{ij}, where the processing time of job i on machine j, an $m \times n \times n$ three-dimensional matrix of setup times s_{ijk}, where the setup time of job j when processed after job k on machine i. Where m is the number of machines and n the number of jobs. Table 12 illustrates an example instance for the problem with $m = 2$ and $n = 3$ then with 6 operations.

Figure 34 illustrates a Gantt chart with an example of the instance solution shown in Table 12. The solution is defined as the sequence of jobs on each machine. Machine 1

Table 12 – Operations, processing times and setup times for open shop example

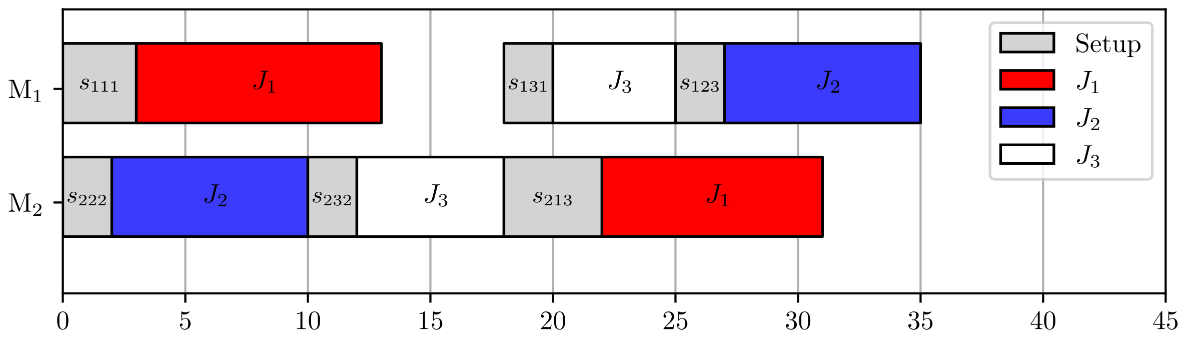
p_{ij} (O_{ij})	J_1	J_2	J_3
M_1	10 (O_{11})	8 (O_{12})	5 (O_{13})
M_2	9 (O_{21})	8 (O_{22})	6 (O_{23})

	M_1			M_2		
s_{ijk}	J_1	J_2	J_3	J_1	J_2	J_3
J_1	3	3	2	2	4	4
J_2	3	2	2	3	2	2
J_3	2	4	2	2	2	3

Source: Authors.

processes jobs 1, 3, and 2 and machine 2 processes jobs 2, 3, and 1.

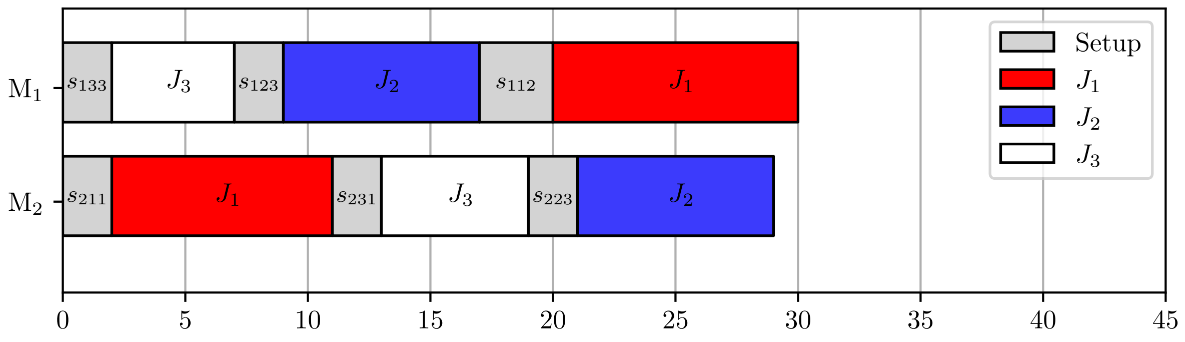
Figure 34 – Gantt chart for scheduling of an example solution (makespan = 35 u.t.)



Source: Authors

The solution in Figure 34 has a makespan of 35 u.t. It is noticeable that the setups are non-anticipatory when the setup of job 3 on machine 1 starts only when job 3 finishes its processing on machine 2. The setup times, when the job j is the first in the machine sequence, is the time of the diagonal of the matrix s_{ijj}. Figure 35 illustrates the optimal solution for the instance, with the smallest makespan of 30 u.t.

Figure 35 – Gantt chart for scheduling of best solution (makespan = 30 u.t.)



Source: Authors

For the solution in Figure 35, there is no idle time between machines, which contributes to minimizing the makespan due to the greater use of resources in the scheduling horizon, unlike the solution in Figure 34, where machine 1 has an idle time of 5 u.t. between the processing of jobs 1 and 3.

6.2.1 Mixed-integer linear programming (MILP)

There are several ways to modeling the OSSPST problem using mathematical programming, the main ones differ in relation to the way of defining the decision variables, with notation of three types: positional notation, sequential notation and time-indexed notation. In the study by Naderi *et al.* (2011b) it is illustrated that models with sequential notation perform better in OSSP problems due to their smaller number of variables and constraints, when compared to positional and time-indexed notation. Therefore, we used the sequential notation of the developed MILP model.

The mathematical model in the equations (6.1) - (6.14) is an adaptation of the model by Naderi *et al.* (2011a) for the OSSPST with non-anticipatory setups. The decision sets, parameters and variables are defined below:

Indices and sets:

$i, l \in M$: indices for machines $\{1, \ldots, m\}$.

$j, k \in N$: indices for jobs $\{1, \ldots, n\}$.

$k \in N_d$: indices for jobs with dummy job $\{0, 1, \ldots, n\}$.

Parameters:

p_{ij}: processing time of job j in machine i.

s_{ijk}: setup time of job j processed immediately after job k in machine i (note: the setup time of the first job j in the sequence of machine i is s_{ij0} with dummy job 0).

M: a large positive number.

Decision variables:

C_{ij}: completion time of operation O_{ij}.

C_{max}: the makespan of the schedule.

Y_{jik}: 1 if $O_{i,j}$ is processed immediately after $O_{i,k}$, and 0 otherwise; $k \neq j$.

X_{jil}: 1 if $O_{i,j}$ is processed not necessarily immediately after $O_{l,j}$, and 0 otherwise; $i < m$ and $l > i$.

The MILP model for the OSSPST is presented below:

minimize

$$C_{max} \tag{6.1}$$

subject to

$$\sum_{k \in N_d : k \neq j} Y_{jik} = 1, \qquad \forall i \in M, j \in N \quad (6.2)$$

$$\sum_{j \in N : j \neq k} Y_{jik} \leq 1, \qquad \forall i \in M, k \in N_d, k > 0 \quad (6.3)$$

$$\sum_{j \in N} Y_{ji0} = 1, \qquad \forall i \in M \quad (6.4)$$

$$Y_{jik} + Y_{kij} \leq 1, \qquad \forall i \in M, j \in N, k \in N_d, j < n, k > j \quad (6.5)$$

$$C_{ij} \geq C_{ik} + p_{ij} + s_{ijk} - M(1 - Y_{jik}), \qquad \forall i \in M, j \in N, k \in N_d, k \neq j \quad (6.6)$$

$$C_{ij} \geq C_{lj} + p_{ij} + \sum_{k \in N_d : k \neq j} (Y_{jik} \times s_{ijk}) - M(1 - X_{jil}), \qquad \forall i, l \in M, j \in N, i < m, l > i \quad (6.7)$$

$$C_{lj} \geq C_{ij} + p_{lj} + \sum_{k \in N_d : k \neq j} (Y_{jlk} \times s_{ljk}) - M \times X_{jil}, \qquad \forall i, l \in M, j \in N, i < m, l > i \quad (6.8)$$

$$C_{i,0} = 0, \qquad \forall i \in M \quad (6.9)$$

$$C_{max} \geq C_{ij}, \qquad \forall i \in M, j \in N \quad (6.10)$$

$$Y_{jik} \in \{0, 1\}, \qquad \forall i \in M, j \in N, k \in N_d, k \neq j \quad (6.11)$$

$$X_{jil} \in \{0, 1\}, \qquad \forall i, l \in M, j \in N, i < m, l > i \quad (6.12)$$

$$C_{ij} \in \mathbb{R}^+ \quad , \qquad \forall i \in M, j \in N_d \quad (6.13)$$

$$C_{max} \in \mathbb{R}^+, \qquad (6.14)$$

Expression (6.1) illustrates the objective function of the problem, constraint (6.2) indicates that each job must be processed only once on each machine, having only one predecessor. Constraint (6.3) forces that each job can be succeeded by up to one job in the sequence of each machine. Constraint (6.4) indicates that the dummy job (index 0) has only one successor on each machine. Constraint (6.5) prohibits each job from being both the successor and predecessor of another job on each machine.

Regarding operation completion times, constraint (6.6) indicates that the completion time of the O_{ij} operation, if the O_{ij} operation is processed after the O_{ik} operation, must be greater than the completion time of the O_{ik} operation plus processing and setup time. Constraint (6.7) indicates that the completion time of job j on the next machine i should be the job completion time on the previous machine l plus the processing and setup time, considering the previous operation of machine i. Constraint (6.8) illustrates the inverse case of constraint (6.7) with the next operation of job j taking place on machine l. Constraints (6.9) and (6.10) indicate the completion time of the dummy job and the total duration of the schedule (makespan), respectively.

Finally, expressions (6.11) - (6.14) illustrate the domain of the model's decision variables, where X_{jil} and Y_{jik} are the integer and binary type and C_{ij} and C_{max} of the real type with values greater than or equal to zero.

6.2.2 Constraint programming (CP)

Among recent emerging techniques applied in exact approaches, such as decomposition and constraint programming methods for scheduling problems (NISHI; HIRANAKA, 2013; HONG; CHOU; LEE, 2019; CHALESHTARTI *et al.*, 2020), the constraint programming has achieved good performance for OSSP (MALAPERT *et al.*, 2012; MEJÍA; YURASZECK, 2020). Constraint programming is a relatively recent paradigm for solving combinatorial optimization problems, especially for complex problems that cannot be easily modelled with integer linear equations and production scheduling problems (ROSSI; BEEK; WALSH, 2006). CP initially emerged in the field of artificial intelligence, but has achieved good results when applied to production scheduling problems (PINEDO, 2016).

The OSSPST problem can be modelled through constraint programming using the same sets and parameters of the MILP model and two types of decision variables: interval and sequence. Interval variables represent an operation to be processed from a job on a machine, with start, end and duration times of the operation. Sequence decision variables, on the other hand, cluster several interval variables into a set, such as the processing sequence of jobs on a machine, for example (LABORIE, 2018). Among the main competitive solvers for constraint scheduling, we use IBM's CP Optimizer, which has been obtaining competitive results for production scheduling problems. (LABORIE; GODARD, 2007; LUNARDI *et al.*, 2020).

In production scheduling problems, with CP Optimizer models, it is common to use transition matrices together with noOverlap constraints to represent the parameter of sequence-dependent setup times (LABORIE *et al.*, 2018). However, the setups represented by transition matrices can only be of the anticipatory type and it is not possible to represent the initial setup time when the machine processes the first job in the sequence. Therefore, as the premises required for the proposed OSSPST are not met, adaptations in the CP model that use transition matrices are necessary.

Next, we show an adaptation of the CP Optimizer model proposed by Mejía and Yuraszeck (2020) for the OSSPST problem, considering non-anticipatory setups and initial setup times for the machines' first job.

Indices and sets:

$i \in M$: index for machines $\{1, \ldots, m\}$.

$j \in N$: index for jobs $\{1, \ldots, n\}$.

Parameters:

p_{ij}: process time of job j in machine i.

s_{ijk}: setup time of job j processed immediately after job k in machine i (note: the setup time of the first job j in the sequence of machine i is s_{ijj}).

Decision variables:

x_{ij}: an interval variable for processing the operation of job j in machine i.

y_{ij}: an interval variable for the setup of job j in machine i.

Γ_i: a sequence variable with order of y_{ij} and x_{ij} intervals variables in machine i with each type of interval variable is the number of job j.

The CP model for the OSSPST is presented below:

minimize

$$\max_{i \in M : j \in N} endOf\left(x_{ij}\right) \tag{6.15}$$

subject to

$$noOverlap\left(\Gamma_i\right), \qquad \forall i \in M \tag{6.16}$$

$$noOverlap\left(\left[y_{ij}, x_{ij}\right]_{i \in M}\right), \qquad \forall j \in N \tag{6.17}$$

$$endAtStart\left(y_{ij}, x_{ij}\right), \qquad \forall i \in M, j \in N \tag{6.18}$$

$$sizeOf\left(y_{ij}\right) = s\left[i\right]\left[j\right]\left[typeOfPrev\left(\Gamma_i, y_{ij}, j\right)\right], \qquad \forall i \in M, j \in N \tag{6.19}$$

$$\text{interval } x_{ij}, \quad \text{size} = p_{ij}, \qquad \forall i \in M, j \in N \tag{6.20}$$

$$\text{interval } y_{ij}, \qquad \forall i \in M, j \in N \tag{6.21}$$

$$\text{sequence } \Gamma_i, \quad \text{on } \left[y_{ij}, x_{ij}\right]_{j \in N}, \quad \text{types } \left[j, j\right]_{j \in N}, \qquad \forall i \in M \tag{6.22}$$

Equation (6.15) illustrates the objective function of the problem of minimizing the maximum completion time of all operations O_{ik} (makespan). Constraint (6.16) indicates that only one job can be processed at a time on machine i. Constraint (6.17) assumes that the same job j cannot be processed by more than one machine at the same time.

Constraint (6.18) forces the processing of job j on machine i to start only after its setup. Constraint (6.19), on the other hand, forces the duration of the variable y_{ij} to be equal to the setup time on machine i of job j processed immediately after the last job present on the machine. The index of the last job processed on machine i is obtained using the typeOfPrev expression, which returns the type of the last variable processed in the Γ_i sequence before the y_{ij} operation. When the y_{ij} variable is the first in the sequence, the third argument of the expression is returned, in this case it is the value j. Therefore, in this case, the expression returns the setup time of job j when it is the first in the sequence of machine i, according to Table 12, for example, it is setup time s_{ijj}.

Finally, constraints (6.20) - (6.22) indicate the domain of the variables. x_{ij} is an interval variable that has duration p_{ij}, y_{ij} is an interval variable and Γ_i is a variable that holds the sequence of setups and operations processed on machine i, all with type j referring to the job that is part of the operation or setup.

Analyzing the model of the equations (6.15) - (6.22), it can be observed that properties of the OSSPST problem can be exploited to improve the CP model, such as the fact that setups are non-anticipatory. Setup time plus processing time can be clustered together when it know the last job processed on the machine.

This feature can be modelled using the logical constraints present in the CP Optimizer, thus developing a new CP model for the problem, whereas variables x and y can be clustered into a single interval variable, thus reducing the number of decision and constraint variables.

Next, we show a new CP Optimizer model proposed in this work for the OSSPST problem, considering a single interval decision variable. The sets and parameters are the same as the previous model.

Decision variables:

z_{ij}: a unique interval variable for setup and processing the operation of job j in machine i.

Υ_i: a sequence variable with order of z_{ij} intervals variables in machine i with each type of interval variable is the number of job j.

The new constraint programming model with unique interval variable (CP-IV) for the OSSPST is presented below:

minimize

$$\max_{i\in M:j\in N} \mathit{endOf}\left(z_{ij}\right) \tag{6.23}$$

subject to

$$\mathit{noOverlap}\left(\Upsilon_i\right), \qquad \forall i \in M \tag{6.24}$$

$$\mathit{noOverlap}\left(\left[z_{ij}\right]_{i\in M}\right), \qquad \forall j \in N \tag{6.25}$$

$$\mathit{sizeOf}\left(z_{ij}\right) = s\left[i\right]\left[j\right]\left[\mathit{typeOfPrev}\left(\Upsilon_i, z_{ij}, j\right)\right] + p_{ij}, \quad \forall i \in M, j \in N \tag{6.26}$$

$$\text{interval } z_{ij}, \qquad \forall i \in M, j \in N \tag{6.27}$$

$$\text{sequence } \Upsilon_i, \quad \text{on} \left[z_{ij}\right]_{j\in N}, \quad \text{types} \left[j\right]_{j\in N}, \qquad \forall i \in M \tag{6.28}$$

Equation (6.23) illustrates the objective function of the problem of minimizing the completion time of all setup operations and processing z_{ij} (makespan). Constraints (6.24) and (6.25) are similar to the constraints of the previous CP model, indicating that a machine only processes one job at a time and a job can only be processed by one machine at a time.

Constraint (6.26) forces the duration of the variable z_{ij} to be equal to the setup time on machine i of job j processed immediately after the last job present on the machine plus the processing time p_{ij}. That is, it is the total time that job j occupies machine i. Finally, (6.27) and (6.28) constraints indicate the domain of the variables. z_{ij} is an interval variable and Υ_i is a variable that keeps the sequence of operations (setups + processing) executed on machine i, all with type j referring to the operation's job.

6.2.3 Discussion

This subsection compares the complexity of the constraints, the number of decision variables and the number of constraints for each of the exact approaches presented. We compares the main components of OSSPST with the different ways of modelling the problem.

The MILP model presented in equations (6.1) - (6.14) uses sequential notation for decision variables, but due to problem domain characteristics, such as non-overlapping constraints for machines and jobs and the presence of the non-anticipatory setup, the model has an expressive constraint complexity. The constraint with the largest size is the one found in equations (6.6) and (6.7) with the worst case complexity of $\mathcal{O}\left(mn^2 + m^2n\right)$. The MILP model has altogether $mn\left(\frac{1}{2} + \frac{3}{2}n + m\right) + n$ constraints and $n\left(m+1\right) + mn^2 + \frac{1}{2}mn\left(m-1\right)$ decision variables, of which $n\left(m+1\right)$ are real and $mn^2 + \frac{1}{2}mn\left(m-1\right)$ are binary integers (NADERI *et al.*, 2011a).

The CP model uses logical constraints for the problem and applies heuristic techniques to reduce the search space for the solution. Furthermore, CP has a greater ease in finding feasible solutions, due to the exploration of the combinatorial problem domain (ROSSI; BEEK; WALSH, 2006). This results in a greater facility for modelling combinatorial problems such as production scheduling, mainly due to the possibility of using logical constraints, which reduces the amount of decision variables and model constraints.

We presented the initial CP model in equations (6.15) - (6.22) which uses the CP Optimizer solver notation for sequencing problems, the constraint with the largest size was presented in equations (6.18) and (6.19) with the worst case complexity of only $\mathcal{O}\left(nm\right)$. The CP model has a total of $m\left(10n+1\right) + n + 4$ constraints and $m\left(2n+1\right) + n$ decision variables, all variables are discrete and of the interval or sequence type. Therefore, due to the CP modelling properties and forms, it has a smaller number of constraints and decision variables than MILP, which can contribute to a better performance in the OSSPST solution.

If the problem have anticipatory setups, it was possible to suppress the terms $\sum_{k \in N_d : k \neq j}(Y_{jik} \times s_{ijk})$ from the constraints (6.7) and (6.8) of the MILP model, reducing the size from the problem to the anticipatory setup times case. However, in the case of the present research study, properties can be explored for the reduction of the problem size with the CP-IV model that will be compared with the other exact strategies in the results analysis section.

The CP-IV model, presented in equations (6.23) - (6.28), uses the CP Optimizer solver notation for scheduling problems, the model seeks to reduce the amount of decision variables and constraints using OSSPST properties such as non-anticipatory setups. The constraint with the largest size is the one present in the equations (6.26) with worst case complexity of $\mathcal{O}(mn)$. The CP-IV model has altogether $m(9n+1)+n+3$ constraints and $m(n+1)+n$ decision variables. The Table 13 illustrates a comparison between the key characteristics of exact approaches, for several different instance sizes. With m the number of machines and n the number of jobs.

Table 13 – Comparison of MILP and CP formulations with examples of instances sizes for OSSPST

Instances sets		MILP			CP		CP-IV	
m	n	# integer variables	# continuous variables	# constraints	# integer variables	# constraints	# integer variables	# constraints
3	3	36	12	75	24	100	**15**	90
4	4	88	20	172	40	172	**24**	**155**
5	5	175	30	330	60	264	**35**	**238**
6	6	306	42	564	84	376	**48**	**339**
7	7	490	56	889	112	508	**63**	**458**
8	8	736	72	1320	144	660	**80**	**595**
9	9	1053	90	1872	180	832	**99**	**750**
10	10	1450	110	2560	220	1024	**120**	**923**
15	15	4950	240	8565	480	2284	**255**	**2058**
20	20	11800	420	20220	840	4044	**440**	**3643**

Source: Authors.

Analyzing the Table 13, it can be observed that with the increase in the size of the problem, the decision variables and restrictions of the MILP models have a significant increase when compared to the variables and restrictions of the CP models. The CP-IV model presented the smallest amounts of decision variables and constraints for most of the presented problem sizes. However, a smaller model will not necessarily perform better than the others, therefore all exact methods will be performed in the testing and results analysis section.

6.3 Proposed algorithm

The adaptive large neighborhood search (ALNS) method is a meta-heuristic based on the large neighborhood search (LNS) method proposed by Shaw (1998). LNS is an optimization method that iteratively performs a process of relaxation and re-optimization of solutions. Both methods have several applications in combinatorial optimization problems, including using hybridization with exact approaches such as integer linear programming and constraint programming (PISINGER; ROPKE, 2007; LI; NEGENBORN; LODEWIJKS, 2017; HOJABRI *et al.*, 2018; HE; WEERDT; YORKE-SMITH, 2019; HÀ *et al.*, 2020). LNS performs a process that comprises two phases: one of destruction or relaxation of the solution and another phase of construction or re-optimization of the solution.

The destruction phase involves removing or destroying a portion of the solution, making it a partial solution to the problem, such as removing a number of jobs from the solution. The relaxation phase, to be more effective, can incorporate properties or characteristics of the problem domain, which can vary depending on the computational representation of the problem-solution (SHAW, 1998).

The construction or re-optimization phase consists of using an exact method to transform the partial solution generated by the destruction phase into a viable solution to the problem. For this construction, constraints programming methods can be used. The technique can receive a partial solution as a warm-start and promote considerable improvements in the solution, such as a local search step (LABORIE *et al.*, 2018).

The LNS uses only one destruction method during its execution. The main improvement of the ALNS is the use of multiple destruction methods along with the iterations; these methods are chosen adaptively, selecting with a higher priority the destruction methods that generate better solutions. Furthermore, using multiple destruction methods, we can explore various properties of the problem (HE; WEERDT; YORKE-SMITH, 2019; HÀ *et al.*, 2020).

ALNS has several recent applications with competitive results in routing and container loading problems (LI; CHEN; HUO, 2022; FRIEDRICH; ELBERT, 2022). However, there are not many applications of ALNS in scheduling problems, more precisely in the open shop and in problems with setup times or costs with hybridization of CP models (AHMADIAN *et al.*, 2021a; ALLAHVERDI, 2015) which is a substantial gap to be solved with the present research.

In addition, ALNS has obtained competitive results when compared to traditional optimization methods in scheduling problems (COTA *et al.*, 2019; LI *et al.*, 2021) due to its adaptive nature of solution destruction and construction. Traditional methods such as LNS and IGA use only single destruction and reconstruction operators throughout their iterations, and ALNS changes the type of operators used adaptively during the execution, using the destruction and construction operators that return the best results. Therefore, ALNS was used as the primary proposed optimization method.

The following subsections explain the development process of the matheuristic ALNS-CP for the OSSPST problem.

6.3.1 Solution representation

For the OSSP, several ways of presenting solutions for heuristic and meta-heuristic methods have already been presented in the literature. Three different coding schemes were the most recurrent: the blocks operations on the critical path, the rank matrix, and the permutation list (ANAND; PANNEERSELVAM, 2016).

The blocks operations on the critical path can be used exclusively for the makespan minimization objective, introduced by Liaw (1999). The rank matrix coding was initially proposed by Bräsel, Tautenhahn and Werner (1993). The matrix represents a data structure where each row is the sequence of operations for a given job on machines and each column represents the sequence of jobs on each machine.

The permutation list encoding consists in a vector with the sequence of operations (that is, each job and machine pair is assigned an index number so that a solution is represented by a sequence). Clearly, the permutation list encoding scheme is much simpler, but its main disadvantage is its redundancy, as different sequences can, in fact, represent the same scheduling for the open shop. Naderi *et al.* (2010) present four theorems to drastically reduce redundancies in permutation list encoding.

The chosen form to represent the solution in the ALNS-CP algorithm is the representation by permutation list. Figure 36 illustrates an example of the permutation list representation on an instance of size 2 x 3 (2 machines and 3 jobs).

Figure 36 – An illustrative example for the permutation list encoding representation

Sequence representation:	1	2	3	4	5	6
Operation representation:	O_{11}	O_{12}	O_{13}	O_{21}	O_{22}	O_{23}
Machines:	1	1	1	2	2	2
Jobs:	1	2	3	1	2	3
Sequence solution:	3	4	2	6	1	5
Decoded solution:	O_{13}	O_{21}	O_{12}	O_{23}	O_{11}	O_{22}

Source: Authors

The figure 36 illustrates the optimal solution for a problem with six operations, presented in Figure 35, with a makespan of 30 u.t. The solution is decoded by ordering the operations according to the sequence present in the permutation list solution. Problem domain characteristics such as the theorems proposed by Naderi *et al.* (2010) will be incorporated in the solution's destruction phase so that only operations that will actually change the makespan of the solution will be removed, for they will be re-inserted in the permutation list to generate a new solution.

6.3.2 Procedure for destroy solutions

We proposes four different strategies for the destruction of solutions, which are the following: random removal, job removal, machine removal and idleness removal operators.

Random removal operator consists of removing a percentage of operations from the solution, randomly, not using any more specific criteria. The number of solutions to be removed is $b_r \times m \times n$, where b_r is the percentage of operations that will be removed and $m \times n$ is the size of the list of operations. The remaining operations constitute a partial solution to the problem ready for the solution construction phase.

Job removal operator consists of selecting a job at random and then removing all operations that use that job from the solution. In order for the process of removing and inserting operations does not produce redundant solutions, the selected operations need to be from the same job (NADERI *et al.*, 2010). The job removal operator uses this concept of the problem domain, which applies to the representation of permutation list solutions.

Machine removal operator consists of randomly selecting a machine and then removing all operations that use that machine from the solution. This removal process also uses another feature of the problem domain, similar to the feature present in the job removal operator. The characteristic is that, in order not to produce redundant solutions, it is necessary that the selected operations are from the same machine (NADERI *et al.*, 2010).

For the job and machine removal operators, a redundant solution is a solution that even after removal and reinsertion of operations it continues with the same makespan (NADERI *et al.*, 2010).

Another important characteristic of the open shop domain is the idleness of operations, considered in the heuristics proposed by Abreu *et al.* (2020). The idleness of an operation can be defined as the time required to wait for job j to be processed on machine i of operation O_{ij}. Equation (6.29) illustrates the idle calculation of an O_{ij} operation. Φ_{ij} represents the idleness generated by the insertion of job j on machine i. Job e was the last job processed on machine i, so the setup of operation O_{ij} is s_{ije}. M and J store the accumulated processing times for each machine i and job j is stored in M_i and J_j, respectively, and both are updated every time when a new operation is inserted.

$$\Phi_{ij} = \begin{cases} J_j - M_i + s_{ije}, & \text{if } J_j > M_i \\ s_{ije}, & \text{otherwise} \end{cases} \tag{6.29}$$

An open shop solution with little idleness between operations can result in a smaller makespan. Therefore, this concept was used in the idle removal operator. This removal operator calculates the idleness of all operations in the sequence; the $b_d \times n \times m$ operations with the highest idleness are removed from the solution, where, where b_d is the percentage of operations to be removed and $m \times n$ is the size of the list of operations.

ALNS-CP has a total of four removal operators for the destruction phase, the choice of which operator to use can be very costly due to the high number of parameters. The standard ALNS has a mechanism for choosing the removal operator based on the performance of each one along the algorithm's iterations.

The choice mechanism consists of a weight to be used to calculate the selection probability of each operator, during the execution of the ALNS-CP. We illustrates in equation 6.30 the calculation of updating the weight of an operator, in an iteration of the algorithm. The mechanism is based on the method proposed by Hojabri *et al.* (2018), adapted to the OSSPST problem with destruction operators considering the solution representation of the OSSPST.

$$w_k = \gamma \cdot w_k + (1-\gamma) \cdot \frac{\pi_k}{\eta_k} \tag{6.30}$$

In equation (6.30), w_k is the weight of an operator k to be randomly selected during an ALNS-CP iteration. π_k is the number of times using the k operator improved the correct solution, η_k is the number of times the operator was selected and γ is a degree of importance that varies between 0 and 1. When γ is close to 1, the weight does not change so much along the iterations, whereas when γ is close to 0 the most important weight update is the recent success rate of its use.

All weights for each removal operator are initially set to 1 so that each operator has an equal chance of selection. The selection probability of an operator k is given by the relative weight of the operator $\left(\frac{w_k}{\sum_{i=1}^{T} w_i}\right)$, where T is the total number of removal operators. This adaptive mechanism means operators that improve the quality of the algorithm solutions may have a greater chance of being selected during ALNS-CP iterations.

The percentage parameters of the random and idle removal operators b_r and b_d, respectively, and the degree of importance γ need to be defined for the OSSPST problem. In subsection 6.3.5, we illustrate the method used for the configuration of all parameters used in the ALNS-CP.

6.3.3 Initial solution procedure

Priority rules for OSSP are largely based on LPT algorithms that sort operations in descending order. However, these approaches do not explore important characteristics of the problem domain, such as idleness between operations. Therefore, the constructive heuristic minimal idleness heuristic (MIH) proposed by Abreu *et al.* (2020) was implemented to create a quality initial solution for ALNS-CP.

MIH allocates in the solution the machine-job operation that returns the minimum idleness in the machine related to the operation. The idleness indicator can be calculated by the accumulated processing times for jobs and machines in the production system throughout the execution of the operations in the scheduling sequence. Equation (6.29) presents the procedure for calculating idleness in subsection 6.3.2.

According to equation (6.29), if the cumulative time for a given job in the system is greater than the accumulated time for a given machine, it means that the machine will wait until the job is finished, and consequently, it can be allocated to the current machine. On the other hand, if the cumulative time of this job is lower than the cumulative time of a given machine, this job was already processed in another machine, and its processing in the current machine will not result in idleness (ABREU *et al.*, 2020). Figure 11 shows the complete pseudocode for MIH where p is the processing time matrix, m is the number of machines, and n is the number of jobs.

```
   Data: $p$, $m$, $n$
   Result: A sequence $\mathit{\Pi} := (O_{11}, O_{12}, ..., O_{mn})$
 1 $\mathit{\Pi} \leftarrow \emptyset$;
 2 $M \leftarrow$ list with time cumulative in each machine;
 3 $J \leftarrow$ list with time cumulative in each job;
 4 $\Omega_k \leftarrow$ list with the jobs allocated to machine $k$, $\forall k \in \{1, ..., m\}$;
 5 while $\|\mathit{\Pi}\| < n \times m$ do
 6     machine $k \leftarrow \underset{i \in \{1,...,m\}}{\text{argmin}}\ M_i$;        // index from machine finishing first
 7     job $j \leftarrow \underset{j \in \{1,...,n\}, j \notin \Omega_k}{\text{argmin}}\ \Phi_{kj}$;  // the best job not allocated in machine k,
        through the MIH rule
 8     $\mathit{\Pi} \leftarrow \mathit{\Pi} \cup \{O_{kj}\}$;
 9     $\Omega_k \leftarrow \Omega_k \cup \{j\}$;
10     update $J$ and $M$ with time of $O_{kj}$ operation;
11 end
```

Figure 37 – Pseudocode of the minimal idleness heuristic

Lines 1-4 start the parameters and data structure of the algorithm. In line 5, there is the main while loop of the algorithm. The algorithm runs while any operations are not inserted in solution $\mathit{\Pi}$. Line 6 gets the index k of the machine released more early, and line 7 gets the index j of the job that has not yet been inserted in the machine k, and that will result in the minimum idleness Φ_{kj}. The selected operation O_{kj} is inserted into the solution in line 8. Line 9 inserts the index of the selected job j in the list of jobs allocated in machine k to prevent the job from being selected again. Finally, line 10 updates the accumulated processing times in the production system for operation of job j and machine k.

6.3.4 Complete pseudo code for hybrid ALNS-CP

Figure 38 illustrates the complete pseudo-code of the ALNS-CP algorithm. The algorithm has three main phases: initial solution construction, destruction, and reconstruction phases. Then, the algorithm iteratively repeats the solution destruction and reconstruction phase, and when the stopping criterion is satisfied, the best solution found ($\mathit{\Pi}_{best}$) is returned.

The ALNS-CP algorithm has four main parameters: the percentage of operations of the random and idle removal operators b_r and b_d, respectively; the degree of importance γ; the set of removal operators $\mathcal{Q}$; and the time limit TL of the CP model in the reconstruction phase. Finally, the algorithm needs the data of the problem instance: the processing time matrix p and the setup time matrix s.

The proposed algorithm also has a search intensification mechanism for when the iterative process does not return solutions better than the current solution over several iterations. The algorithm performs the solution reconstruction phase by running the CP model with the initial solution $\Pi_{partial}$ with a time limit longer than the default TL parameter. The function $REPAIR\_SOLUTION$ executes the CP model and, in the end, converts the interval variables to the sequence-of-operations solution representation of the figure 36. The function $DESTROY\_SOLUTION$ applies the selected removal operator.

```
   Data: $p$, $s$, $b_r$, $b_d$, $\mathcal{Q}$, $\gamma$, $TL$
   Result: A sequence $\Pi_{best} := \{O_{11}, O_{12}, ..., O_{mn}\}$
1  $\Pi_{current} \leftarrow$ a solution ordered by a constructive heuristic MIH;
2  $\Pi_{best} \leftarrow \Pi_{current}$;
3  makespan_best $\leftarrow makespan(\Pi_{best}, p, s)$;
4  $w \leftarrow$ initial weights for each removal operators with value 1.0;
5  $stableit \leftarrow 0$;                 // number of iterations without improvement
6  while the stopping criterion is not satisfied do
7      select randomly a removal operator $\mathcal{Q}_k$ with weight $w_k$;
8      $\Pi_{partial} \leftarrow DESTROY\_SOLUTION(\mathcal{Q}_k, \Pi_{best}, b_r, b_d)$;
9      if $stableit < 10$ then
10         $\Pi_{current} \leftarrow REPAIR\_SOLUTION(\Pi_{partial}, time\_limit = TL)$;
11         $\Pi_{current} \leftarrow local\_search(\Pi_{current})$;        // apply 2-opt local search
12     else
13         $\Pi_{current} \leftarrow REPAIR\_SOLUTION(\Pi_{partial}, time\_limit = 5 \times TL)$;
14     end
15     if $makespan(\Pi_{current}, p, s) < makespan\_best$ then
16         $\Pi_{best} \leftarrow \Pi_{current}$;
17         makespan_best $\leftarrow makespan(\Pi_{best}, p, s)$;
18         $stableit \leftarrow 0$;
19     else
20         $stableit \leftarrow stableit + 1$;
21     end
22     update $\pi_k$ and $\eta_k$ parameters of selected removal operator $\mathcal{Q}_k$;
23     $w_k = \gamma \cdot w_k + (1 - \gamma) \cdot \frac{\pi_k}{\eta_k}$;
24 end
```

Figure 38 – Pseudocode of the adaptive large neighborhood search with constraint programming

Lines 1-5 start the parameters and data structure of the algorithm. The algorithm obtains the initial solution by the MIH constructive heuristic proposed by (ABREU *et al.*, 2020). In line 6, there is the main while loop of the algorithm. The algorithm runs while the execution time limit is not exceeded. In line 7, the removal operator is chosen randomly with the probability vector created by the weights w of each removal operator. The higher the weight of an operator, the more probable it is to be selected. In line 8, the ALNS-CP executes the selected removal operator, producing the partial solution $\Pi_{partial}$.

In lines 9-14, the algorithm executes the procedure to repair the partial solution. If the number of iterations without improvement is less than 10, the function $REPAIR\_SOLUTION$ is executed, which uses the CP model to reconstruct the partial solution and improve the solution with iterations of the CP Optimizer solver, with a time limit of TL seconds. After the reconstruction phase, the ALNS-CP applies a fast 2-opt best-improvement local search in line 11, with the redundant solution filters proposed by Naderi *et al.* (2010).

If the number of iterations without improvement is greater than or equal to 10, ALNS-CP performs the solution repair procedure with a five-fold time limit. The CP optimizer, when run from a partial or complete initial solution, can improve the solution in the solver iterations (LABORIE; GODARD, 2007). Furthermore, with the more intensive iterations of the CP optimizer, the ALNS-CP algorithm may escape possible local optima solutions.

In lines 15-21, if the newly obtained solution $\Pi_{current}$ is better than the best solution found Π_{best}, the best solution is updated, the algorithm reset the unimproved iterations counter to zero, and start the ALNS cycle of destruction and reconstruction phases again. Otherwise, the interaction counter with no improvement is updated, and the algorithm returns to the destruction and reconstruction phases with the current best solution. The solution acceptance criterion is of the hill-climbing type (RUIZ; STÜTZLE, 2008).

Finally, lines 22 and 23 update the success rate parameter and the weight w_k of the removal operator used. After the end of iterations, the algorithm returns the best solution found Π_{best}.

6.3.5 Parameter's optimization

Researchers frequently use the IRACE package to find the best parameters for optimization algorithms like metaheuristics (LÓPEZ-IBÁNEZ *et al.*, 2016). Therefore, we used the IRACE package to obtain the best parameters for the proposed ALNS-CP algorithm. Table 14 illustrates the values of each parameter considered, and in the Selected value column is the best result obtained from each parameter in the IRACE tests. IRACE provides the test result information for a set of parameters, and each parameter has a range of options available. For example, in Table 14, the IRACE package recommends adopting intermediate values of parameters b_r and b_d, the smallest value for parameter γ, and the largest available value for parameter TL.

During the execution of IRACE, the algorithm iteratively updates the way the parameters are sampled, aiming to use the region of values of these parameters that improve the optimization algorithms' performance. Therefore, the sampling frequency of values of each parameter can provide essential insights into the behavior of the optimization algorithm when these parameters are changed (LÓPEZ-IBÁNEZ *et al.*, 2016). Figure 39

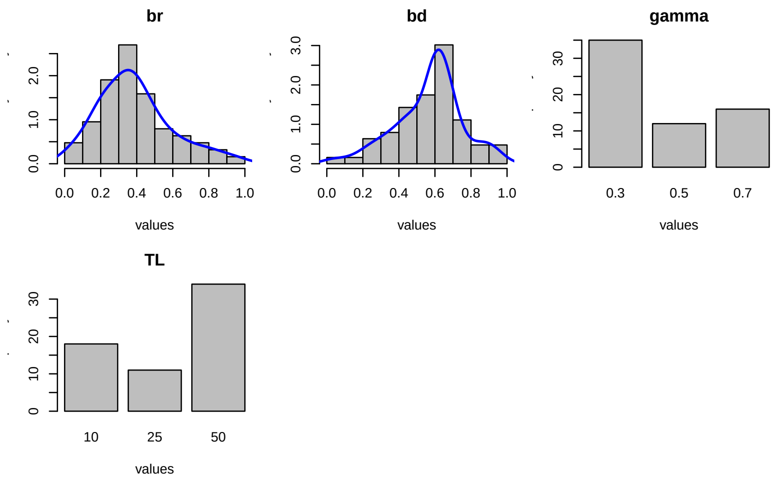
Table 14 – IRACE parameter range settings and resulting values

Parameter	IRACE name	Range	Selected value
b_r	br	$[0,1]$	0.4052
b_d	bd	$[0,1]$	0.6588
γ	gamma	$\{0.3, 0.5, 0.7\}$	0.3
TL	TL	$\{10, 25, 50\}$	50

Source: Authors.

shows the sampling frequency of the values of each parameter for the developed ALNS-CP algorithm.

Figure 39 – Parameters sampling frequency for developed method.



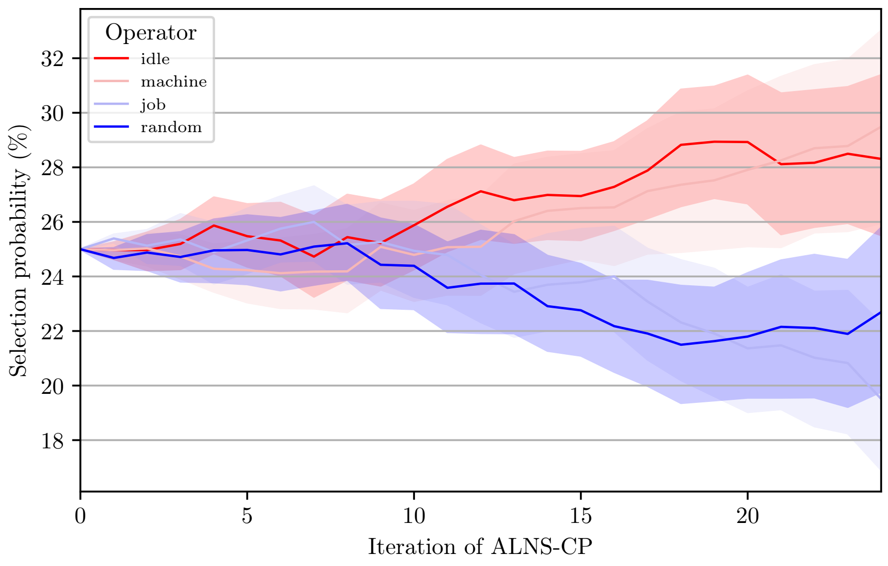
Source: Authors

In classical LNS algorithms, only one removal operator is used (SHAW, 1998). However, ALNS-CP applies four different removal operators that consider important characteristics of the problem domain, such as the operation's idleness. Furthermore, the ALNS-CP selects, adaptively, the removal operators that result in better solutions throughout the algorithm iterations.

We perform a simulation to analyze the evolution of the selection probability of the removal operators over the iterations of ALNS-CP. The simulation considers an example instance of size 49 (seven machines and seven jobs), with the stopping criterion equal to 24 iterations $\left(\left\lfloor \frac{m \times n}{2} \right\rfloor\right)$, where $m \times n$ is the size of the problem. The execution of ALNS-CP in 24 iterations was performed 100 times to generate a confidence interval (95% of confidence) for the selection probability of the removal operator in each iteration. Figure 40 illustrates the evolution of the selection probability for each of the four removal operators for each iteration performed.

Figure 40 shows that all the four removal operators start with an equal probability of selection of 25% and get different probability selection values throughout the iterations. For

Figure 40 – An example of selection probability and confidence interval (alpha=0.05) of removal operators by iterations in a instance with size 49.



Source: Authors

this tested instance, the idle and machine removal operators have the highest probability of selection at the end of iterations, which may indicate that the selection of these operators implies an improvement of the current ALNS-CP solution. For these two operators, the π_k term of the equation (6.30) increased its value due to obtaining better solutions than the current one. The increment of π_k increases the weight w of the operators and, consequently, increases the selection probability of these operations in the iterations of ALNS-CP.

In addition, the selection of the random and job removal operators decreases throughout the iterations, which may indicate that these operators are not selected or when ALNS-CP uses them, the operators do not imply an improvement of the current solution. For these two operators, the π_k term of the equation (6.30) did not get a significant increase in its value, due to not obtaining solutions better than the current one, this causes a stagnation in the weight w of the operators and, consequently, reduces the selection probability of these operations throughout the iterations of ALNS-CP, due to other better operators increase their weight w and consequently their selection probability.

6.4 Computational experience

The section describes the methodology used for the computational experience of the tested methods, the sets of instances, and the performance criteria to compare the algorithms. We divided the comparison between the exact methods and the approximate methods for solving the OSSPST.

6.4.1 Instances sets and performance criteria

We evaluate the proposed exact and approximate techniques using benchmarking problems from the literature. Taillard (1993), Guéret and Prins (1998) and Brucker *et al.* (1997) propose these problems. Then, we use these datasets for testing algorithms for solving classical OSSP, adapted for OSSPST by Abreu *et al.* (2020). The adaptation has similar criteria as the instance generations proposed by Mejía and Yuraszeck (2020), Naderi *et al.* (2011a) and Roshanaei, Esfehani and Zandieh (2010).

For the processing times matrix, in the instances of Guéret and Prins (1998), the processing times are random values, uniformly distributed between 1 and 1000. The problem sizes are $n, m \in \{3, 4, 5, 6, 7, 8, 9, 10\}$. For each problem class, 10 instances were generated, totaling 80. In Taillard (1993) instances, the processing times are random values, uniformly distributed between 1 and 100. The problem sizes are $n, m \in \{4, 5, 7, 10, 15, 20\}$. For each problem class, 10 instances were generated, totaling 60. In Brucker *et al.* (1997) instances, the processing times are random values uniformly distributed between 1 and 500. The problem sizes are $n, m \in \{3, 4, 5, 6, 7, 8\}$, totaling 60 instances. In summary, we test 192 instances for OSSPST.

We divided each set of instances into two groups; the low type instances have the setup distribution with uniform numbers between [1, 499]. The high type instances have a setup distribution between [500, 999]. We used the set of instances from Abreu *et al.* (2020) for benchmarking against various algorithms already tested for Open Shop and OSSPST problems. In addition, the set of instances is robust by not only use the adaptation of Taillard instances, considered easy to solve by the literature due to having the optimal solution equal to the lower bound of the problem (MALAPERT *et al.*, 2012; TAILLARD, 1993; ANAND; PANNEERSELVAM, 2016). Therefore, Abreu *et al.* (2020) considers the three main sets of OSSP instances and further divides the setups into low and high instances to verify the performance of the algorithms with different setup distributions.

The first indicator used to measure the efficacy of the computational experiments is the relative percentage deviation (RPD) between the solution obtained by the method and the lower bound of the instance. We calculate the average RPD result for each solving method tested on the instance sets. Equation (6.31) illustrates the average relative percentage deviation indicator (ARPD) for a method h, where sol_{kh} is the value of instance k obtained by method h, LB_k is the lower bound for instance k. K and H are the numbers of instances and methods, respectively.

$$ARPD_h = \frac{1}{K} \sum_{k=1}^{K} \frac{sol_{kh} - LB_k}{LB_k} \cdot 100, \forall h \in \{1, \ldots, H\} \tag{6.31}$$

Mejía and Yuraszeck (2020)) propose the lower bound used in the test of OSSPT, which considers the setup times in the calculation, to have a tighter result close to the optimal integer solution. Equation 6.32 describes the lower bound LB_k for an instance k, where p_{ij} is the processing times matrix, and TSP_i is the optimal tour for a traveling salesman problem with the setup matrix of machine i. The distances from the source node to the other nodes are the setup times of each job when the machine starts (diagonal of the setup matrix), and the distances from the other nodes are the setup time from one job to the other. M and N are the sets of machine jobs, respectively, and K is the number of instances.

$$LB_k = \max\left\{\max_{\forall i \in M}\left\{\sum_{j\in N} p_{ij} + TSP_i\right\}, \max_{\forall j\in N}\left\{\sum_{i\in M} p_{ij}\right\}\right\}, \forall k \in \{1,\ldots,K\} \tag{6.32}$$

To measure the efficiency of the computational experiments, the indicator used is the average execution time of the algorithms for the set of tested instances. For example, equation (6.33) illustrates the average computational time (ACT) indicator for a method h, where CT_{kh} is the value of the computational time of method h when executing instance k. K and H are the numbers of instances and methods, respectively.

$$ACT_k = \frac{1}{K}\sum_{k=1}^{K} CT_{kh}, \forall h \in \{1,\ldots,H\} \tag{6.33}$$

We implement all the proposed methods and benchmarking algorithms in the Python 3.7 language (https://www.python.org/). The MILP model and the CP model were implemented in the IBM ILOG CPLEX 12.10 solver. The computational experiments were executed on a virtual machine with Intel® Xeon(R) CPU E5-2660 2.20GHz and 8GB of RAM. All source codes, instances, results, computational times, and statistical tests are available at http://repositorio.uspdigital.usp.br/handle/item/446.

6.4.2 Computational results for exact approaches

For the exact models, we adopt a time limit of 1800 seconds in the computational tests. The tested methods are listed below.

- Mixed Integer Linear Programming (MILP): the proposed mathematical programming model given by the Equations (6.1)-(6.14).
- Constraint Programming (CP): the CP model proposed by Mejía and Yuraszeck (2020) and adapted for OSSPST with non-anticipatory setup times given by the Equations (6.15)-(6.22).
- Constraint Programming with unique Interval Variable (CP-IV): the new constraint programming model with unique interval variable for the OSSPST given by the Equations (6.23)-(6.28).

Laborie *et al.* (2018) indicate CP Optimizer has several different search strategies fixed before the solver execution: Auto strategy (CP Auto), which has a hybridization of exact algorithms with Self-Adapting Large-Neighborhood Search metaheuristics. Depth-first strategy (CP DF) is a tree search algorithm. Restart strategy (CP RS), where the constructive search is restarted at defined intervals and oriented to obtain the optimal solution quickly. Finally, the MultPoint strategy (CP MP) creates a combined set of solutions to produce better solutions over iterations, similar to evolutionary strategies. The Auto and Restart strategies have proof of optimally of the solutions, while the Depth-first and Multipoint strategies do not have due to their heuristic nature in solver iterations.

Therefore, for each of the two CP models developed, the four search strategies were applied, totaling eight different CP methods. MILP was run using the branch and bound (B&B) method for integer linear programming problems. Table 15 illustrates the ARPD for each exact method evaluated on each set of instances (GP - Guéret and Prins, Tai - Taillard and Bru - Bruckner). The results are grouped by instance size ($m \times n$) and by setup type. The last four rows provide a statistical summary of the computational results: the best ARPD value (Min), the average ARPD (Average), the worst ARPD value (Max), as well as the number of unsolved instances for each exact proposed method. The results with the symbols *** indicate that the method used did not find any feasible solution in the computational time provided.

Table 15 – ARPD results of exact methods in all sets of instances

Instances sets			MILP	CP				CP-IV			
Group	Size	Setup	B&B	Auto	DF	RS	MP	Auto	DF	RS	MP
GP	9	Low	**17.25**	**17.25**	**17.25**	**17.25**	**17.25**	**17.25**	**17.25**	**17.25**	**17.25**
		High	**9.44**	**9.44**	**9.44**	**9.44**	**9.44**	**9.44**	**9.44**	**9.44**	**9.44**
	16	Low	**14.01**	**14.01**	15.33	**14.01**	**14.01**	**14.01**	15.19	**14.01**	**14.01**
		High	**13.60**	**13.60**	**13.60**	**13.60**	**13.60**	**13.60**	**13.60**	**13.60**	**13.60**
	25	Low	**20.22**	**20.22**	22.44	**20.22**	21.35	**20.22**	22.28	**20.22**	21.28
		High	**12.80**	**12.80**	12.86	**12.80**	12.84	**12.80**	12.86	**12.80**	12.84
	36	Low	**13.81**	14.73	21.31	17.33	25.04	**13.81**	19.85	**13.81**	18.40
		High	14.60	16.07	20.19	16.43	17.56	**13.27**	16.01	13.36	17.35
	49	Low	22.32	27.51	51.88	28.98	40.09	**19.40**	40.83	22.52	33.92
		High	26.93	21.62	27.88	21.65	22.09	**17.74**	24.92	19.43	21.48
	64	Low	27.70	22.64	60.30	26.53	36.49	**17.61**	46.47	20.79	36.52
		High	33.10	22.72	29.00	22.45	22.71	**21.62**	25.63	22.16	24.56
	81	Low	40.20	41.08	86.96	39.57	59.00	**37.73**	74.67	40.60	60.94
		High	***	26.15	32.67	**25.53**	27.36	25.65	30.55	25.90	25.63
	100	Low	47.45	37.93	89.35	46.21	75.42	**37.06**	88.56	39.68	72.44
		High	***	29.44	36.25	30.12	32.01	**27.74**	36.68	29.54	30.46
Tai	16	Low	**15.74**	**15.74**	17.85	**15.74**	**15.74**	**15.74**	17.85	**15.74**	**15.74**
		High	**8.22**	**8.22**	**8.22**	**8.22**	**8.22**	**8.22**	**8.22**	**8.22**	**8.22**
	25	Low	**16.74**	**16.74**	20.36	**16.74**	16.93	**16.74**	20.19	**16.74**	18.13
		High	**7.05**	**7.05**	**7.05**	**7.05**	**7.05**	**7.05**	**7.05**	**7.05**	7.46
	49	Low	**14.81**	23.57	66.99	26.87	40.33	17.63	53.46	19.57	37.05
		High	24.70	16.30	25.51	17.04	20.53	**15.37**	23.11	15.55	18.95
	100	Low	43.61	44.78	141.14	49.69	100.41	**41.68**	121.02	43.85	99.71

Continued on next page

Table 15 continued from previous page

Instances sets			MILP	CP				CP-IV			
Group	Size	Setup	B&B	Auto	DF	RS	MP	Auto	DF	RS	MP
		High	***	26.80	36.90	29.28	29.14	**24.40**	33.45	24.89	29.47
	225	Low	110.06	92.11	202.53	103.87	165.94	**81.13**	196.17	85.94	167.40
		High	***	37.90	47.24	39.28	41.45	**35.41**	47.06	38.36	41.00
	400	Low	209.90	151.20	244.80	174.55	212.41	**136.67**	239.24	148.73	214.46
		High	***	41.58	50.09	42.97	44.16	**41.28**	48.85	42.11	43.60
Bru	9	Low	**14.64**	**14.64**	**14.64**	**14.64**	**14.64**	**14.64**	**14.64**	**14.64**	**14.64**
		High	**6.22**	**6.22**	7.20	**6.22**	**6.22**	**6.22**	6.31	**6.22**	**6.22**
	16	Low	**21.96**	**21.96**	25.75	**21.96**	**21.96**	**21.96**	25.75	**21.96**	**21.96**
		High	**17.08**	**17.08**	**17.08**	**17.08**	**17.08**	**17.08**	**17.08**	**17.08**	**17.08**
	25	Low	**17.82**	**17.82**	20.92	**17.82**	18.14	**17.82**	20.92	**17.82**	19.64
		High	**17.23**	**17.23**	20.70	**17.23**	19.71	**17.23**	20.70	**17.23**	19.12
	36	Low	**15.74**	17.09	38.52	17.68	32.29	**15.74**	28.43	**15.74**	26.31
		High	**11.35**	12.55	27.53	14.06	23.36	**11.35**	20.88	**11.35**	19.39
	49	Low	**17.91**	25.29	52.96	24.89	40.32	19.63	49.14	17.97	36.07
		High	**21.62**	32.55	62.03	32.46	54.41	24.73	61.32	23.96	41.95
	64	Low	**24.06**	34.66	100.56	30.50	66.12	24.51	81.32	25.76	48.28
		High	**25.01**	35.50	91.51	35.35	61.87	29.50	77.15	29.56	55.33
Min			**6.22**	**6.22**	7.05	**6.22**	**6.22**	**6.22**	6.31	**6.22**	**6.22**
Average			27.85	27.05	47.37	28.58	38.12	**24.52**	43.35	25.53	36.43
Max			209.90	151.20	244.80	174.55	212.41	**136.67**	239.24	148.73	214.46
Unsolved instances			32	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**

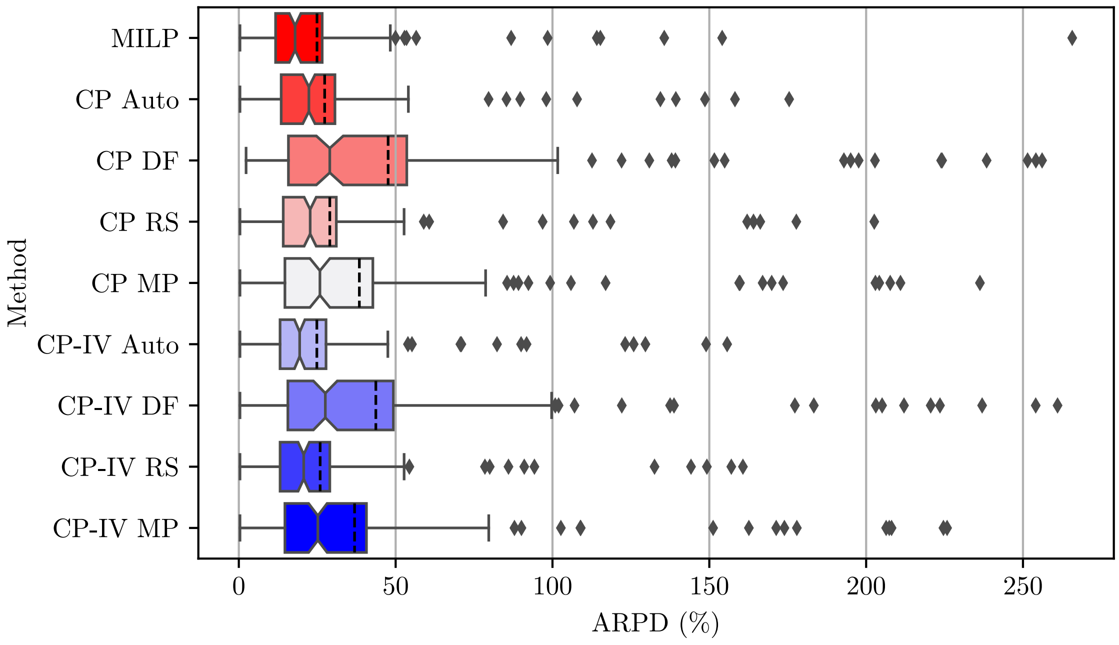
Analyzing Table 15, only the MILP method had unsolved instances. The method could not solve instances with sizes greater than 81 with the setup type high for the Gueret and Prins and Taillard sets. All CP methods were able to find feasible solutions for all instances in the computational time provided. The CP-IV methods outperformed the CP methods, showing that the improvement of the CP model implies a better quality of solutions due to the reduction of constraints and decision variables.

MILP had the best performance on Brucker instances. CP-IV Auto had the best average result and the best result on Gueret and Prins and Taillard instances. Overall, the RS strategy obtained reasonable solutions on Taillard and Gueret and Prins instances and solved more instances than the DF and MP strategies with the best ARPD. The MILP method obtained lower ARPD on instances with low type setups, while the CP models obtained lower ARPD on instances with high type setups.

The CP DF and CP-IV DF methods obtained the worst results among the search methods tested. Figure 41 illustrates a boxplot of the distribution of the ARPD obtained by the proposed methods on all tested instances. The dotted line is the average result for each method.

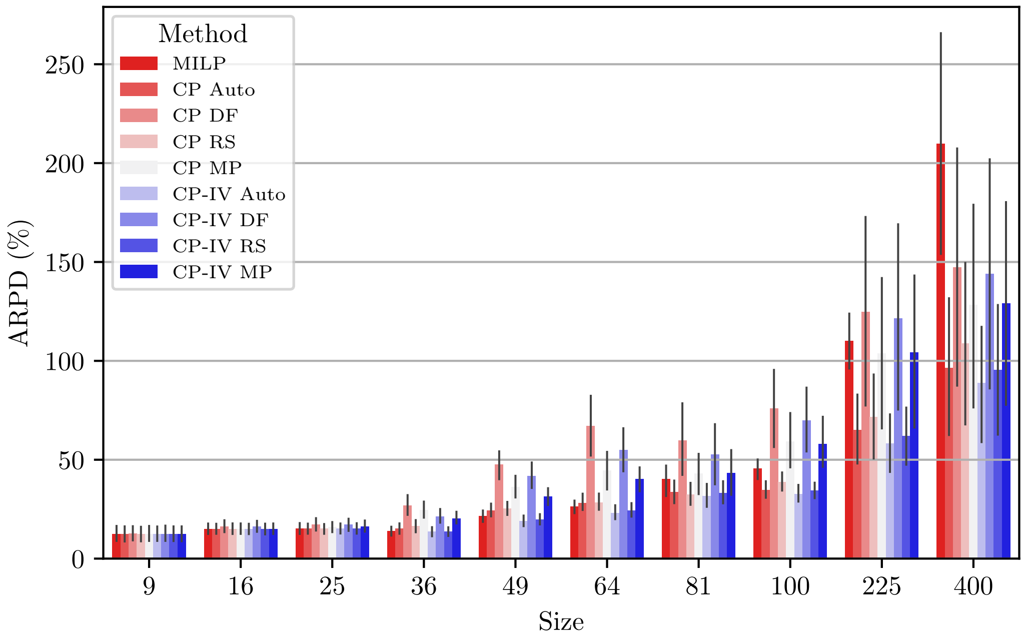
Overall, CP-IV Auto obtained the best results, followed by CP-IV RS and CP RS. In addition, the CP-IV Auto has the lowest median and mean among the methods that can solve all instances. For a better comparison, Figure 42 illustrates each proposed method's ARPD, grouped by instance size.

Figure 41 – ARPD for all exact methods in all sets of instances.



Source: Authors

Figure 42 – ARPD and confidence interval ($\alpha = 0.05$) for all exact methods in each instance size.



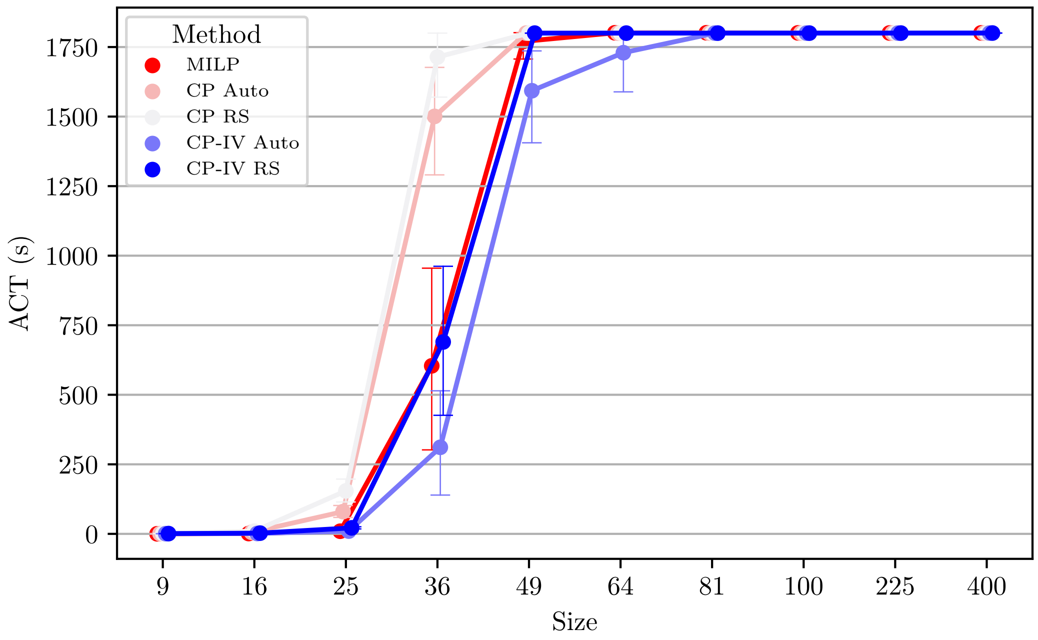
Source: Authors

Concerning figure 42, MILP performs well up to instance size 64, with the performance loss gradually increasing with increasing instance size, showing the difficulty of solving the OSSPST by exact integer linear programming methods. The CP-IV method obtained the best performance compared to the other methods in most of the instance sizes. In instance sizes larger than 100, all exact methods obtained unsatisfactory results, with ARPD greater than 50%.

Therefore, with the increase of complexity of the problem, considering initial machine setups and asymmetric setups (setup time from job j to job k may be different from the setup time from job k to job j on the same machine), solving the problem becomes very costly for instances larger than 100, with no exact method obtaining good results with the time limit used.

Regarding computational times, Figure 43 illustrates the average computational times obtained for each method for each instance size tested. The CP and CP-VI methods with the MP and DF type search strategies have no optimally proof and executed the search for 1800 seconds for all instances, so they were not considered in the visualization.

Figure 43 – ACT and confidence interval ($\alpha = 0.05$) for all exact methods in each instance size.



Source: Authors

The MILP model from instances of size 64 has significant computational times, denoting the difficulty of solving the problem by exact methods on large instances. CP-IV has the best computational times because it is a model with fewer decision variables and constraints, obtaining competitive times mainly for instances of sizes 36 to 64. The CP Auto and CP RS methods, on the other hand, obtained the highest computational cost, especially for instances of sizes 25 to 36. CP methods get the largest variability in computational times. All the methods, starting from the instances of size 81, reached the 1800 seconds time limit.

Since none of the exact strategies could obtain satisfactory results, especially for the larger instances, we tested the meta-heuristic described in Section 6.3 with the hybridization of ALNS with CP as a solution building mechanism. We used the CP search strategy that obtained the best preliminary results (CP-IV Auto) with an initial solution by the MIH constructive heuristic proposed by Abreu *et al.* (2020).

6.4.3 Computational results for approximation approaches

For the computational testing of the approximation approaches, we test several competitive heuristics and metaheuristics from the OSSP and OSSPST literature, adapted for the variant considered in the research, for comparison with ALNS-CP. In summary, we test seven approximate approaches for OSSPST.

- Minimal Idleness Heuristic (MIH): a constructive heuristic proposed by Abreu *et al.* (2020).
- Genetic Algorithm with restart procedure and direct decoding mechanism of solution (GA (D)): the best genetic algorithm proposed by Abreu *et al.* (2020) for OSSPST.
- Electromagnetic Heuristic (EH): a metaheuristic proposed by Naderi *et al.* (2011a) for OSSPST.
- A self-tuning variable neighborhood search algorithm (VNS): a metaheuristic proposed by Mejía and Yuraszeck (2020) for OSSPST with anticipatory setup times and adapted to non-anticipatory case.
- An innovative biased random key genetic algorithm with implicit path-relinking (BRKGA): a metaheuristic for general scheduling problems, proposed by Andrade *et al.* (2019) and adapted to OSSPST.
- Competitive Hybrid Genetic Algorithm (HGA): a competitive HGA for OSSP proposed by Ahmadizar and Farahani (2012) and adapted to OSSPST in this study.
- A hybrid adaptive large neighborhood search with constraint programming (ALNS-CP): the method proposed by us.

We added the six benchmarking approaches as a complementary reference (the lower bound is the basis for the comparisons). In addition, we tested the MIH method. Since ALNS-CP uses MIH in the initial solution as a hybrid approach, it is possible to evaluate the improvement provided by ALNS-CP on the solution generated by MIH.

The metaheuristics HGA, EH, GA (D), and VNS are the most recent approximate approaches applied for OSSP and OSSPST, and there are no articles that test on ensembles and perform robust comparisons with all these metaheuristics (ALLAHVERDI, 2015;

AHMADIAN *et al.*, 2021a). Such tests constitute an important contribution to the present research. Therefore, by comparing ALNS-CP with these techniques, it is possible to verify the improvement in the solution quality compared to the best techniques in the literature for the problem, being possible to analyze the impact of hybridizing CP models with the ALNS technique on the OSSPST problem.

The comparison of ALNS-CP with BRKGA validates if the proposed solution method for OSSPST can obtain better solutions than general and competitive techniques for the scheduling problem. Therefore, the comparison is a validation of the proposed method.

For the HGA, VNS, and BRKGA techniques, an adaptation of the objective function was necessary to consider the makespan of OSSPST, considering explicit and non-anticipatory setup times, according to the illustration of the makespan of the Gantt charts 34 and 35.

For the HGA, EH, GA (D), BRKGA, and VNS the same parameters used by Ahmadizar and Farahani (2012), Naderi *et al.* (2011a), Abreu *et al.* (2020), Andrade *et al.* (2019), and Mejía and Yuraszeck (2020) respectively, were considered in the tests. In HGA we use the parameters: population size 200, crossover rate 0.9, mutation rate 0.2, local optimization heuristic rate 0.2, maximum number of iterations of the local optimization heuristic 200 and q 0.9. In EH we use the parameters: population size 10, initial temperature 20, CT 2, and FN 20. In GA (D) we use the parameters: population size 100, mutation ratio 0.05, restart ratio 0.5, and initial temperature 100. In BRKGA we use the parameters: $|\mathcal{P}|$ 4500, $\mathcal{P}_e\%$ 11, $\mathcal{P}_m\%$ 35, π_t 2, π_e 3 and Φ e^{-r}. Finally, in VNS we use the parameters: a 50 and decoding scheme m-AS.

We implemented all metaheuristics in the same Python 3.7 programming language and adopted the same stopping criteria for all algorithms for a fair comparison. We adopted the stopping criterion of $10 \times m \times n$ iterations per method, which depends on the size of the problem, where m is the number of machines and n is the number of jobs. In addition, we used a run-time limit of 1800 seconds. The required parameters of each benchmarking method were the values used by the respective authors.

Table 16 illustrates the average relative percent deviation (ARPD) for each benchmark method evaluated on each set of instances (GP - Guéret and Prins, Tai - Taillard and Bru - Bruckner). The results are grouped by instance size ($m \times n$) and by setup type. The last four rows provide a statistical summary of the computational results: the best ARPD value (Min), the average ARPD (Average), and the worst ARPD value (Max).

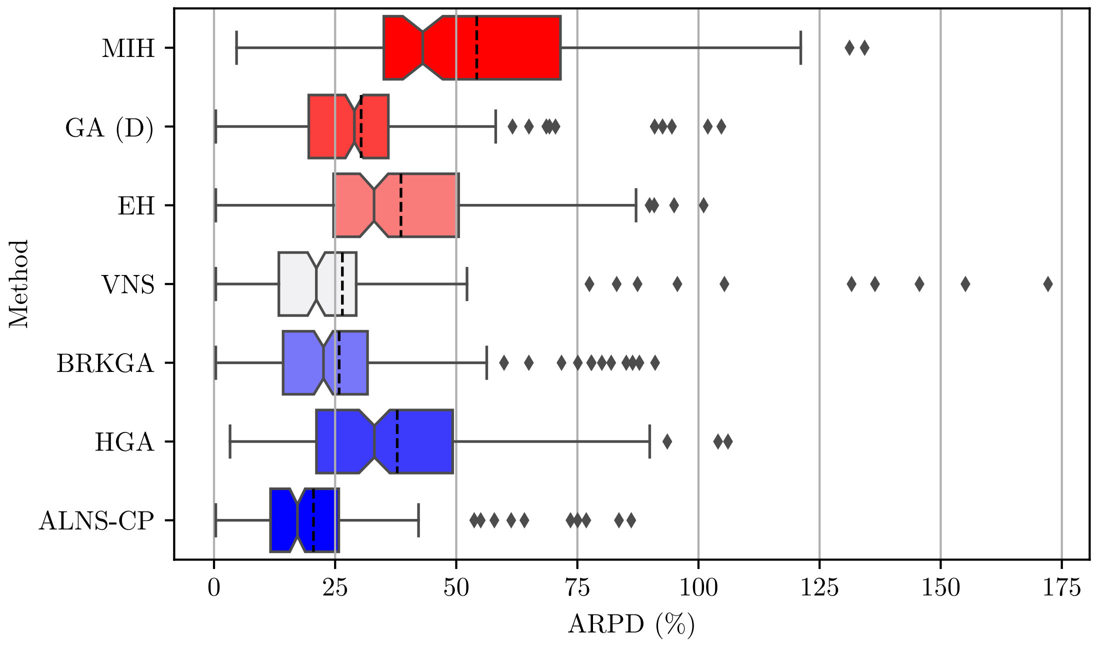
Table 16 – ARPD results of approximation methods in all sets of instances

Instances sets			Approximation approaches						
Group	Size	Setup	MIH	GA (D)	EH	VNS	BRKGA	HGA	ALNS-CP
GP	9	Low	34.65	**17.25**	**17.25**	**17.25**	**17.25**	**17.25**	**17.25**
		High	22.32	**9.44**	**9.44**	**9.44**	**9.44**	**9.44**	**9.44**
	16	Low	36.47	20.89	16.56	**14.01**	**14.01**	21.57	**14.01**
		High	29.45	20.08	16.48	**13.60**	**13.60**	23.65	**13.60**
	25	Low	55.57	30.40	36.34	**20.22**	**20.22**	27.83	**20.22**
		High	35.14	17.38	21.34	**12.80**	14.79	20.02	**12.80**
	36	Low	53.89	22.74	38.18	14.44	16.62	47.51	**13.81**
		High	34.31	19.20	27.16	14.68	20.08	24.53	**13.27**
	49	Low	68.24	31.23	45.36	25.98	25.77	66.38	**18.94**
		High	32.54	21.23	26.56	20.15	23.40	35.83	**16.08**
	64	Low	55.01	28.60	43.19	21.17	20.31	23.29	**13.92**
		High	36.77	27.66	29.26	21.54	27.45	47.94	**19.41**
	81	Low	76.64	38.98	58.52	39.39	37.32	47.23	**30.36**
		High	35.72	29.49	29.23	24.93	30.92	42.09	**22.34**
	100	Low	74.89	40.32	63.87	36.61	47.65	37.52	**30.16**
		High	38.27	31.67	31.85	28.19	32.50	83.10	**25.58**
Tai	16	Low	56.37	20.08	25.04	**15.74**	**15.74**	27.51	**15.74**
		High	18.84	12.29	10.81	**8.22**	**8.22**	14.76	**8.22**
	25	Low	69.79	35.49	42.25	**16.74**	17.67	52.76	**16.74**
		High	36.89	12.39	23.57	**7.05**	9.29	19.91	**7.05**
	49	Low	74.83	30.24	53.24	22.07	21.41	18.90	**14.43**
		High	36.08	18.02	32.11	14.91	22.37	51.96	**9.66**
	100	Low	104.47	49.98	73.12	43.05	55.14	55.09	**28.27**
		High	36.57	30.30	31.36	25.28	31.81	66.79	**19.49**
	225	Low	104.76	65.62	81.74	89.81	81.56	89.38	**58.41**
		High	39.16	33.94	34.71	36.57	35.67	42.34	**33.83**
	400	Low	103.78	96.94	91.99	148.18	81.44	91.09	**79.04**
		High	34.41	44.82	32.94	39.88	32.58	40.86	**31.66**
Bru	9	Low	32.62	17.93	**14.64**	**14.64**	**14.64**	21.92	**14.64**
		High	25.74	9.09	**6.22**	**6.22**	**6.22**	9.29	**6.22**
	16	Low	50.25	28.15	28.35	**21.96**	**21.96**	25.43	**21.96**
		High	52.57	21.68	22.00	**17.08**	**17.08**	27.42	**17.08**
	25	Low	66.42	30.11	36.75	**17.82**	17.93	51.24	**17.82**
		High	56.10	27.84	39.37	**17.23**	**17.23**	53.90	**17.23**
	36	Low	70.85	29.96	46.60	16.79	17.16	31.35	**15.74**
		High	57.80	32.13	44.26	11.86	14.75	15.12	**11.35**
	49	Low	68.63	32.58	67.60	23.79	21.20	22.18	**13.91**
		High	86.81	43.37	62.61	30.96	25.94	25.66	**20.44**
	64	Low	80.73	36.14	67.95	33.05	29.17	26.30	**17.90**
		High	87.91	41.39	66.33	33.88	32.56	33.86	**22.04**
Min			18.84	9.09	**6.22**	**6.22**	**6.22**	9.29	**6.22**
Average			54.31	30.18	38.65	26.18	25.50	37.26	**20.25**
Max			104.76	96.94	91.99	148.18	81.56	91.09	**79.04**

Most methods obtained the best solutions on small instances up to size 9. The matheuristic ALNS-CP obtained the best results for the Min, Average, and Max ARPD indicators, which shows the improvement of the quality of solutions with the hybridization of exact and approximate techniques for OSSPST.

The ALNS-CP, VNS, and BRKGA methods obtained the best results in instances up to 25. From the larger instances of Gueret and Prins, Taillard, and Brucker, the ALNS-CP method obtained the best ARPD results. Also, about table 16, the bold values illustrate the best ARPD results for each instance set. The ALNS-CP method was the one that obtained the best results for each of the tested instance sizes. Figure 44 illustrates a boxplot of the distribution of ARPD obtained by the approximate methods on all tested instances. The dotted line illustrates the average result for each method.

Figure 44 – ARPD for all approximation methods in all sets of instances.

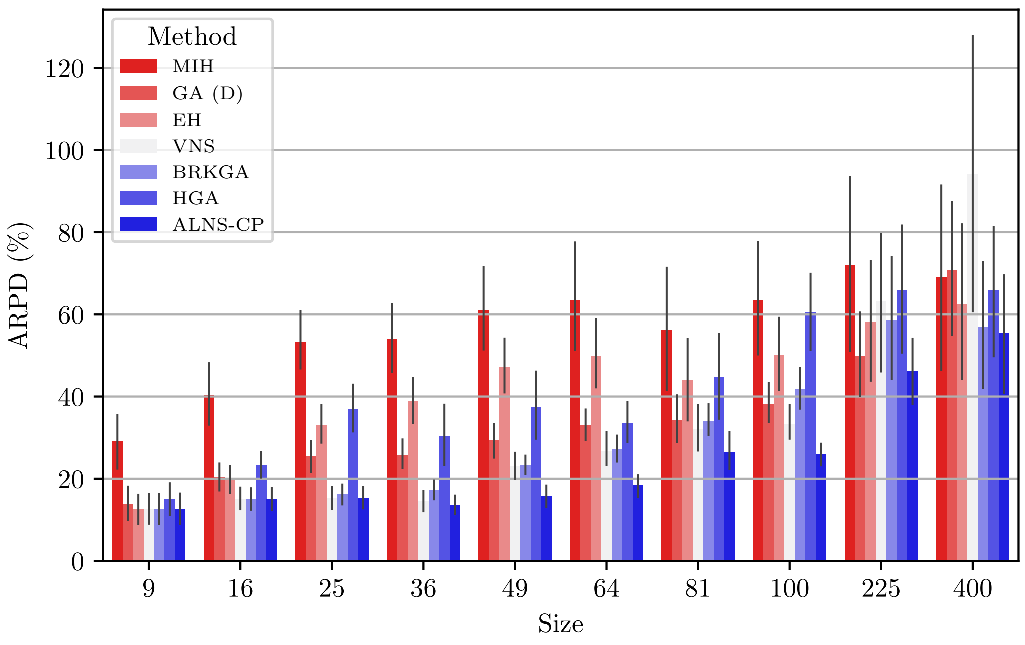


Source: Authors

Overall, ALNS-CP obtained the best results, followed by BRKGA and VNS. Comparing the matheuristic method and the tested metaheuristics, the proposed ALNS-CP math heuristic has the lowest mean, median, and outliers values. For a better comparison, figure 45 illustrates the ARPD of each of the approximate tested methods, grouped by instance size.

From figure 45, MIH performs poorly compared to the other strategies up to instance size 100, with its performance becoming similar to the other methods at instance sizes 225 and 400. The HGA method has good performance up to instance size 81; for instances with larger sizes, the method obtained the worst results among all metaheuristics. The VNS and BRKGA methods had competitive results on most instance sizes, with BRKGA having better results on large instances of sizes 225 and 400. Both methods had ARPD results larger than 50% on the larger instances, which shows that the problem

Figure 45 – ARPD and confidence interval ($\alpha = 0.05$) for all approximation methods in each instance size.



Source: Authors

considering initial machine setups and asymmetric setups has a costly resolution for these metaheuristics.

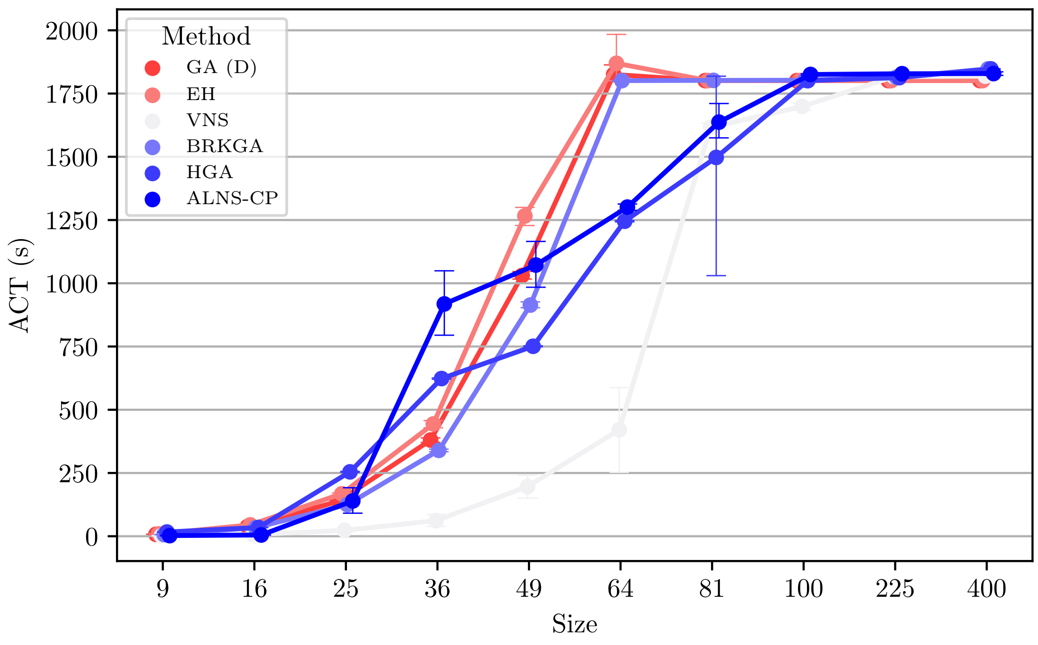
The ALNS-CP method got the best ARPD results in relation to the others methods in most of the instance sizes, with a significant difference mainly for the intermediate instances sizes between 49 to 100, with the confidence interval not crossing with most of the other methods.

Regarding table 16, for instances of size 400 with low setups, all approximate methods obtained results with ARPD greater than 50%. For size 400 with setup type high instances, the approximate methods obtained similar results, with ALNS-CP gets the best ARPD result of 31.66%.

Regarding computational times, Figure 46 illustrates the average computational times obtained for each method for each instance size tested. The MIH method is a constructive heuristic and has a negligible computational time (less than 1 second), so the method is not in the visualization.

The GA (D) and EH methods from instances size 64 have expressive computational times, showing the difficulty of solving larger problems by these approaches. VNS has the best computational times, especially intermediate instance sizes, due to an efficient solution decoding mechanism and perturbation filters on operations (MEJÍA; YURASZECK, 2020)). Therefore, VNS achieved competitive computational times mainly, for instance sizes 36 to

Figure 46 – ACT and confidence interval ($\alpha = 0.05$) for all approximation methods in each instance size.



Source: Authors

64. The ALNS-CP method performed competitive computational times on the instances sizes 64 and 81, obtaining better computational times than the GA (D), EH, and BRKGA methods. All the methods, starting from the instances size 225, reached the 1800 seconds time limit.

It is essential to check whether the differences in ARPD values are statistically significant between the approximate methods tested to validate the results. Since the data are not normally distributed and do not have similar variances, non-parametric tests are recommended (LATORRE *et al.*, 2020). We perform a Kruskal-Wallis test for non-parametric analysis of variance (MONTGOMERY, 2017), the p-value is very close to zero.

Table 17 illustrates the non-parametric Mann-Whitney rank test to see if the differences in ARPD means between the proposed ALNS-CP method and the other benchmarking algorithms are significant. Table 17 shows the difference of ARPD of ALNS-CP compared with other methods (Average diff. (%)). ALNS-CP obtained significantly better results when compared peer-to-peer with all other benchmarking methods.

We applied the ALNS-CP algorithm to the instances of Gueret and Prins, Taillard, and Brucker in the classical OSSP. We perform this test not to develop a new algorithm for the OSSP problem but rather to validate the proposed algorithm on classic instances in the literature, where the best solution is known (SHA; HSU, 2006). Table 18 shows the ARPD and ACT results of the ALNS-CP method for the classical OSSP instances.

Table 17 – Mann-Whitney rank test between ALNS-CP and all other benchmarking methods.

ALNS-CP vs.	Average diff. (%)	Statistic U	p-value
MIH	-33.7	3519	**0.0000**
GA (D)	-9.83	10484	**0.0000**
EH	-18.06	8077.5	**0.0000**
VNS	-5.96	15653	**0.0053**
BRKGA	-5.3	14408	**0.0001**
HGA	-17.29	9049	**0.0000**

Source: Authors.

We added a new stopping criterion in the proposed method in addition to the number of iterations and time limit. The new stopping criterion indicates that the algorithm stops when it found the best solution for the instance for the classic OSSP.

Table 18 – Results of the ALNS-CP on the classical OSSP instances.

Group	ARPD (%)	ACT (s)	Group	ARPD (%)	ACT (s)	Group	ARPD (%)	ACT (s)
GP03	**0.00**	0.46	tai_4x4	**0.00**	1.01	j4	**0.00**	0.68
GP04	**0.00**	2.35	tai_5x5	**0.00**	2.94	j5	**0.00**	2.93
GP05	**0.00**	2.41	tai_7x7	**0.00**	6.01	j6	**0.00**	8.11
GP06	**0.00**	6.60	tai_10x10	**0.00**	6.91	j7	0.02	445.00
GP07	**0.00**	7.56	tai_15x15	**0.00**	2.12	j8	0.35	1185.98
GP08	**0.00**	12.75	tai_20x20	**0.00**	10.25			
GP09	**0.00**	24.53						
GP10	**0.00**	38.85						

Source: Authors.

In Table 18, ALNS-CP found the optimal solutions for all instances of the Gueret and Prins and Taillard sets in competitive computational times with less than 100 seconds. For the Brucker set of instances, the proposed method found the best solutions up to set j6. As for the larger sets of instances, the optimal solutions of some instances were not found, with ALNS-CP reaching either the iteration limit or the computational time limit.

Therefore, analyzing the results reported in Tables 16 and 18 and Figures 45 and 46. VNS and BRKGA methods obtained better solutions than GA (D) and EH metaheuristics and better computational times. ALNS-CP outperformed the tested benchmarking methods MIH, GA (D), EH, VNS, and BRKGA concerning solution quality. For classical OSSP, the proposed method found the optimal solution for most tested instances with admissible computational times. Therefore, ALNS-CP is so far considered a competitive meta-heuristic for OSSPST, with a good trade-off between solution quality and computational cost.

6.5 Conclusion and future studies

In the present paper, we addressed the production scheduling problem in an open shop environment, with sequence-dependent setup times and the objective function of minimizing the total scheduling duration (makespan). Furthermore, we developed a math heuristic hybridizing the adaptive large neighborhood search algorithm with a constraint scheduling model as a local search strategy.

As contributions of the paper: we proposed a new constraint programming scheduling model that considers characteristics of the OSSP problem with non-anticipatory setup times to reduce the number of constraints and variables. As a result, the new CP model got superior performance in solution quality and computational times compared to traditional MILP and CP models. Furthermore, the proposed CP model was hybridized with the ALNS algorithm and obtained competitive performances compared with benchmarking metaheuristics in the literature as EH proposed by Naderi *et al.* (2011a), GA (D) proposed by Abreu *et al.* (2020) and VNS proposed by Mejía and Yuraszeck (2020).

As an extension of this work, we suggests the study of the OSSPST problem considering travel times and resource consumption constraints, which are some of the main parameters present in complex industrial systems. It is also recommended a comparison between exact approaches, applying the constraint programming methods developed from the paper against decomposition approaches, such as Dantzig-Wolfe decomposition, Lagrangian relaxation, and Benders decomposition. Furthermore, different objective functions for the problem can be considered, such as total tardiness and total flow time. Finally, it is possible to easily apply the methods developed in the paper to solve other emerging production scheduling problems, including practical cases in real industries.

7 A NEW TWO-STAGE CONSTRAINT PROGRAMMING APPROACH FOR OPEN SHOP SCHEDULING PROBLEM WITH MACHINE BLOCKING

7.1 Introduction

In the last decades, mechanical and industrial engineering practitioners have paid much attention to production manufacturing scheduling systems. This attention occurs because the optimization of such systems involves cost reduction and planning automation to the companies (BEHNAMIAN; GHOMI, 2016). In general, parallel machines, flow shop, and job shop production environments have received much attention from operational research practitioners. Although the open shop scheduling problem presents theoretical and practical importance, the researchers have paid less attention to this class of problems. Thus there are several research opportunities in this production environment.

The open shop scheduling problem is a production environment in which the jobs are processed in m machines where all the machines can process any of the n jobs, and each machine presents a specific processing time for a given job. There are no fixed routes in the machines, leading to an increase in the search space compared to the other production scheduling environments, such as flow shop or job shop, for example.

The open shop environment arises in several practical situations, such as maintenance, telecommunications, oil industry, plastic molding, chemical, highway construction, food production, medical care services, and teacher-class timetabling (GONZALEZ; SAHNI, 1976; LIN; LEE; PAN, 2008; NADERI *et al.*, 2010; NADERI; NAJAFI; YAZDANI, 2012; ABREU *et al.*, 2020; PALACIOS *et al.*, 2015; ARAÚJO; BONATES; PRATA, 2021; ABREU *et al.*, 2022).

The blocking assumption, also called zero buffer constraints, means that the intermediate storage of materials between adjacent production stages is forbidden because of physical or operational restrictions. This characteristic is typical of many relevant production environments in the industry (HALL; SRISKANDARAJAH, 1996; MOCCELLIN *et al.*, 2018).

The first contribution to the flow shop scheduling problem with machine blocking is presented by Levner (1969). Recently, Miyata and Nagano (2019) presented a review article on the m-machine flow shop scheduling problem considering machine blocking constraints. Mascis and Pacciarelli (2002) firstly addressed the m-machine with blocking considerations. To the best of our knowledge, variants of open shop scheduling problem taking into account machine blocking and makespan minimization as performance measures have not been previously reported in the literature of the last four decades of research on the open shop scheduling problem to minimize the makespan (AHMADIAN *et al.*, 2021b).

This paper explores a variant of the open shop problem considering machine-blocking constraints for makespan minimization as the objective function (OSSPB). Under the state of the art, this variant has not been addressed before, despite its theoretical importance and real-world applications. We can highlight real-world applications, such as plastic molding, chemical and pharmaceutical industries, medical laboratory analysis, and advanced manufacturing environments with just-in-time production systems (HALL; SRISKANDARAJAH, 1996; BAI *et al.*, 2017).

One of the more recent proposed variants of the open shop is the addition of no-wait constraints that have differences from blocking constraints (AHMADIAN *et al.*, 2021b). The jobs must be processed continuously in the available machines without interruption in the no-wait open shop. Therefore, there is no wait time for a job between its processing in two machines. On the other hand, in the open shop with machine blocking, the intermediate storage capacity between machines is considered limited (buffer zero constraints). Consequently, a job finished on a given machine blocks it until the next machine is available.

The main contributions of this paper are listed as follows. First, the study of open shop variant with machine-blocking constraints to fill the gaps in current literature. Second, we propose two mixed-integer linear programming (MILP) models for the variant under study, which can solve small-sized instances optimally, and we propose a new constraint programming (CP) model considering machine-blocking constraints. Third, we propose a two-stage constraint programming model as a new exact method for obtaining high-quality solutions within admissible computational times, including the ability to solve large-sized problems. Finally, we develop a new set of instances with challenging sizes to test the performance of proposed methods compared to benchmarking methods.

The remainder of this paper is organized as follows: in Section 7.2, the literature review is presented; in Section 7.3, the proposed exact methods are presented; in Section 7.4, a two-stage constraint programming algorithm is proposed; in Section 7.5, some results from computational experiments are discussed; finally, in Section 7.6 the final remarks are presented.

7.2 Literature review

To the best of our knowledge, the $O_m|block|C_{max}$ variant has not been previously addressed in the available literature. In this section, some related approaches are reviewed to study similar production environments. Sidney and Sriskandarajah (1999) presented a two-machine no-wait open shop scheduling problem and proposed a heuristic for the makespan minimization. As with no-buffer constraints, the intermediate storage of jobs between production stages is forbidden in the no-wait environment.

Yao, Soewandi and Elmaghraby (2000) presented a two-machine scheduling problem in an open shop with blocking, considering n jobs, two machines and the minimization of the makespan. Four heuristics are presented, with a better performance of the random search algorithm.

Lin, Lee and Pan (2008) presented a multi-processing-stage open shop with movable dedicated machines and no-wait constraints and proposes a mixed integer programming model and a two-phase heuristic. The objective function minimizes the total occupation time for all the processing stages. Naderi, Najafi and Yazdani (2012) presented an open shop with no buffer, considering multiple machines and jobs with the objective function of minimizing the total tardiness. As solutions procedures, a MILP formulation and an electromagnetism-like metaheuristic are developed. Finally, Naderi and Zandieh (2014) studied an open shop in a no-wait environment, taking into consideration multiple machines and jobs and proposing three mixed-integer programming models and two metaheuristics (variable neighborhood search and genetic algorithm). The objective function is the makespan minimization.

Mejía, Caballero-Villalobos and Montoya (2018) studied the m-machine open shop scheduling problem with deadlocking and blocking considerations. An ordinary Petri Net is proposed to model this production environment. Several heuristic functions are developed using the structural properties of the proposed Petri Net.

Due to its similarity to the open shop problem, the blocking job shop problem has a large number of recent contributions that can help the development of new solution methods for the open shop with blocking. Dabah *et al.* (2019) presented a multi-start tabu search approach to reduce infeasible solutions in the search process. Lange and Werner (2019) proposed new neighborhood structures and repair techniques with simulated annealing to blocking job shop with total tardiness minimization. The computational tests highlight the capability of the proposed method to construct feasible schedules of valuable quality, even large-size instances.

Abreu and Nagano (2022) proposed an adaptive large neighborhood search with constraint programming for the open shop with sequence-dependent setup times. The problem differs from the one proposed in this paper since it does not consider machine blocking, considering only sequence-dependent setup times. However, the constraint programming model developed by Abreu and Nagano (2022) outperforms other exact approaches, indicating that a constraint programming model for the variant with machine blocking could obtain competitive performance. Furthermore, the open shop with setup paper proposes a hybridized metaheuristic with a constraint programming model, while the present paper develops a two-stage constraint programming approach as an exact method.

Finally, Mogali, Barbulescu and Smith (2021) presented new primal heuristic updates to improve local search methods with job insertion moves for blocking job shop.

The results showed a significant improvement to the computational efficiency of existing local search procedures. However, according to the review, few exact methods are applied to the problem. Therefore, with a further contribution of the paper, the exact approaches proposed in the manuscript can be adapted for solving the blocking job shop and compared with benchmarking methods.

Table 19 illustrates the main contributions of the literature to the OSSP considering blocking and similar constraints. The authors, year of publication, characteristics of the problem such as the type of setup, solution methods and main research contributions are illustrated.

Table 19 – Summary of the main contributions from the OSSP literature considering blocking and similar constraints.

Author	**Problem characteristics**	**Solution method**	**Contribution**
Sidney and Sriskandarajah (1999)	No-wait	Polinomial time heuristic	The heuristic requires $\mathcal{O}\left(n \log n\right)$ for the two machine case.
Yao, Soewandi and Elmaghraby (2000)	Blocking	RSA	A new meta-heuristic that outperforms classics flowshop-based heuristics.
Lin, Lee and Pan (2008)	Multi-processing-stage andno-wait	MILP and two-phase heuristics	The proposed heuriscs got optimal and near-optimal results in small-sized instances.
Naderi, Najafi and Yazdani (2012)	Blocking	EH	EH outperforms MILP in small and large-sized instances.
Naderi and Zandieh (2014)	No-wait	VNS and GA	A new encoding and decoding shcheme for solutions and GA outperforms all other metaheuristcs in benchmarking instances.
Mejía, Caballero-Villalobos and Montoya (2018)	Blocking	Heuristics with petri net	The method found solutions very close to the lower bound.

Source: Authors.

According to the literature review by Ahmadian *et al.* (2021b), OSSPB is still not well explored, and there is not much work considering buffer zero or machine blocking constraints. Thus, the paper's main contribution further explores the OSSPB by proposing: new efficient exact methods as MILP and CP models as a new hybrid method that incorporates characteristics of CP and two-stages approaches for solving the OSSPB. In addition, another gap pointed out by Ahmadian *et al.* (2021b) is the existence of a few sets of instances for the problem. Thus, we introduced new sets of challenging instances to test proposed solution methods with benchmarking methods of the current literature of OSSP.

7.3 Problem statement, mathematical formulations and properties

This section illustrates a example of instance for the OSSP, new integer programming and constraint programming model and some proprieties of the problem as lower bound and valid inequalities.

7.3.1 An illustrated example

Let n be the number of jobs to be scheduled in a set of m machines. Each job j presents an associated processing time p_{ij} on machine i. In this problem, there is no pre-established production route. Each processing time is associated an operation O_{ij} of job j in machine i that can be processed in any sequence. Furthermore, zero buffer (machine blocking) constraints are considered: the intermediate storage of jobs is not allowed between two adjacent machines. Each machine processes one job at a time and each job can be processed by one machine at a time. Hereinafter, this problem is referred to as the open shop with blocking (OSSPB).

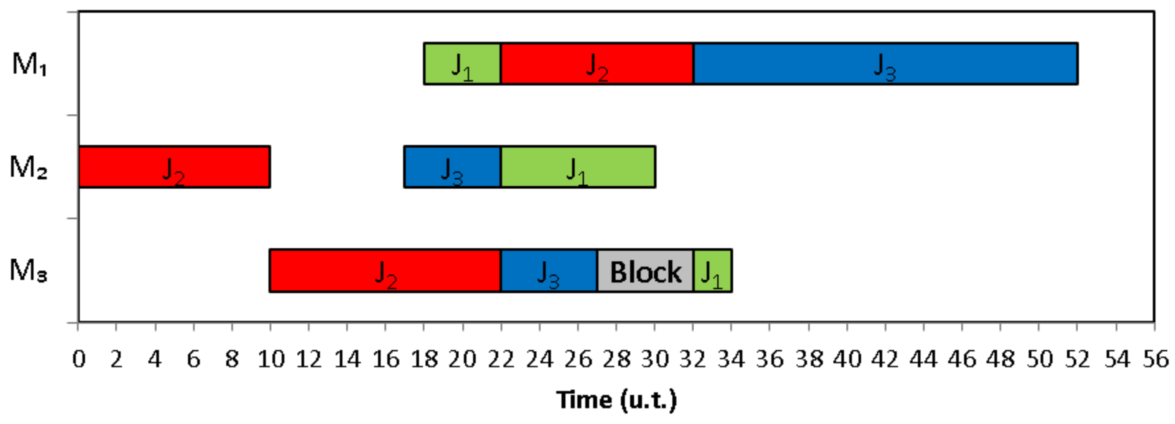
Given these definitions, the problem under study is to find the sequence, the start and end dates of each operation on the machines that minimize the total duration of the schedule or the maximal completion time (makespan or C_{max}). Table 20 describes an instance for the OSSPB with three machines and three jobs, where O_{ij} is the operation of job j in machine i and p_{ij} is the matrix of processing times of job j in machine i. Figure 47 illustrates an Gantt Chart of a feasible solution for the problem under study. For this solution, job 2 is processed on machines 2, 3, and 1; then job 3 is processed on machines 2, 3, and 1; finally, job 1 is processed on machines 1, 2, and 3. Each operation is processed at the earliest possible time to avoid blocking the machines. The presented solution has a makespan of 52 time units (u.t.) with the end of job 3 processing on machine 1.

Table 20 – Processing times per operations for a example of instance

p_{ij} (O_{ij})	J_1	J_2	J_3
M_1	4 (O_{11})	10 (O_{12})	20 (O_{13})
M_2	8 (O_{21})	10 (O_{22})	5 (O_{23})
M_3	2 (O_{31})	12 (O_{32})	5 (O_{33})

Source: Authors.

Figure 47 – Gantt chart for the feasible solution (makespan = 52 u.t.)

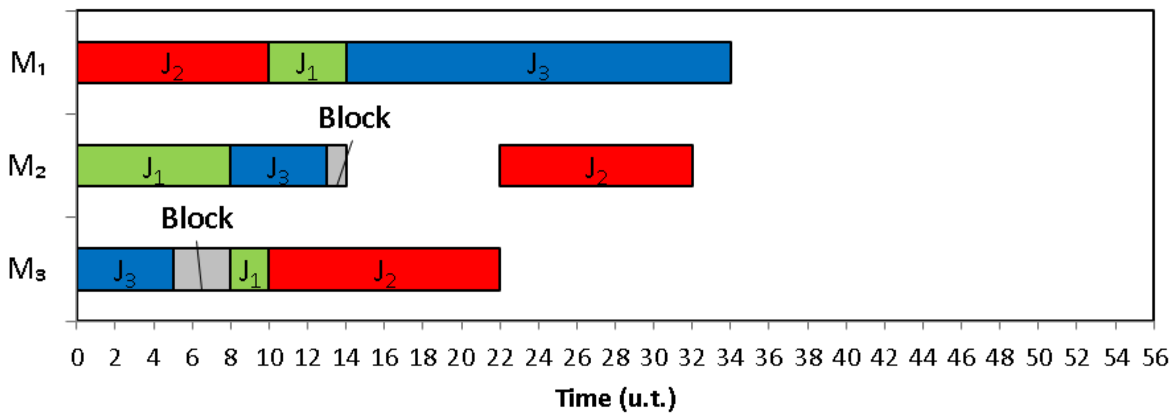


Source: Authors

In Figure 47, we can see the blocking time for machine 3 between the processing of jobs 3 and 1. The blocking time is 5 u.t. due to the waiting time of job 3 on machine 3 before job 3 processing on machine 1. This waiting time blocks machine 3 until the current job leaves it to go to another machine or leave the schedule (when it is the last job to enter the machine).

This blocking delays the start time of job 1 on machine 3, which could have started at time 30 u.t. (when the job processing on machine 2 was finished). However, due to the blocking of machine 3, it was started only at time 32 u.t. Figure 48 illustrates the optimal solution for the example instance, with a makespan of 34 u.t.

Figure 48 – Gantt chart for the optimal solution (makespan = 34 u.t.)



Source: Authors

Figure 48 shows that no occurrence of machine blocking is delaying any operation from starting. For example, the blocking of job 3 on machine 3 ends precisely when the processing of job 1 on machine 3 should be started (after the end of job 1 on machine 2), with no delays occurring. This efficient allocation of the blocks contributes to the reduction of the makespan.

7.3.2 Mixed-integer linear programming models

Although mixed-integer programming models are usually not efficient methods for solving large-sized instances of many production scheduling problems due to their NP-hard nature, we present a mathematical programming model for the problem to assess the quality of the heuristics proposed. A MILP model can provide high-quality or even optimal solutions for small-sized or medium-sized test instances.

Among the main decision variable notations for production scheduling problems, we can highlight the positional and the sequence-based notation (STAFFORD; TSENG; GUPTA, 2005). These two notations have been competitive performances in MILP models of classical open shop problems (NADERI *et al.*, 2011b). Since the problem with makespan blocking, m machines and minimization is new and exact approach models have never been applied, MILP models will be developed exploiting these two main decision variable notations to check which has the best performance.

The first MILP (MILP1) model uses a sequence-based notation for the decision variables. Each one represents a part of a sequence of the operations not necessarily immediately between two machines or two jobs. MILP models with this type of notation has a good performance in classic OSSP (NADERI *et al.*, 2011b). Next, we describe the notation and equations adopted in the developed MILP model.

Indices and sets:

$j \in \mathcal{N}$: index for jobs $\{1,2,...,n\}$.

$k \in \mathcal{N}_d$: index for jobs with dummy job $\{0,1,2,...,n\}$.

$i \in \mathcal{M}$: index for machines $\{1,2,...,m\}$.

$l \in \mathcal{M}_d$: index for machines with dummy machine $\{0, 1,2,...,m\}$.

Parameters:

p_{ij}: processing time of job j in machine i (with dummy $p_{i0} = 0, \forall i \in \mathcal{M}_d$ and $p_{0j} = 0, \forall j \in \mathcal{N}$).

M: a large and positive number.

Decision variables:

C_{max}: makespan.

T_{jl}: starting time of job j in machine l.

X_{il}^{j}: 1 if job j is processed in machine i immediately after machine l, and 0 otherwise; $l \neq i$.

Y_{jk}^{i}: 1 if job j is processed immediately after job k in machine i, and 0 otherwise; $k \neq j$.

The proposed MILP formulation is presented as follows.

minimize

$$C_{max} \tag{7.1}$$

subject to

$$C_{max} \geq T_{ij} + p_{ij}, \qquad \forall i \in \mathcal{M}, j \in \mathcal{N} \tag{7.2}$$

$$T_{0j} = 0, \qquad \forall j \in \mathcal{N} \tag{7.3}$$

$$\sum_{l \in M_d, l \neq i} X_{il}^{j} = 1, \qquad \forall i \in \mathcal{M}, j \in \mathcal{N} \tag{7.4}$$

$$\sum_{k \in N_d, k \neq j} Y_{jk}^{i} = 1, \qquad \forall i \in \mathcal{M}, j \in \mathcal{N} \tag{7.5}$$

$$\sum_{i \in M, i \neq l} X_{il}^{j} \leq 1, \qquad \forall l \in \mathcal{M}, j \in \mathcal{N} \tag{7.6}$$

$$\sum_{j\in N, j\neq k} Y_{jk}^{i} \leq 1, \qquad \forall i \in \mathcal{M}, k \in \mathcal{N} \tag{7.7}$$

$$X_{il}^{j} + X_{li}^{j} \leq 1, \qquad \forall i \in \mathcal{M}, j \in \mathcal{N}, l \in \mathcal{M}_d, l > i \tag{7.8}$$

$$Y_{jk}^{i} + Y_{kj}^{i} \leq 1, \qquad \forall i \in \mathcal{M}, j \in \mathcal{N}, k \in \mathcal{N}_d, k > j \tag{7.9}$$

$$\sum_{i\in M} X_{i0}^{j} = 1, \qquad \forall j \in \mathcal{N} \tag{7.10}$$

$$\sum_{j\in N} Y_{j0}^{i} = 1, \qquad \forall i \in \mathcal{M} \tag{7.11}$$

$$T_{ij} \geq T_{lj} + p_{lj} - M(1 - X_{il}^{j}), \qquad \forall j \in \mathcal{N}, i, l \in \mathcal{M}, l \neq i \tag{7.12}$$

$$T_{ij} \geq T_{ik} + p_{ik} - M(1 - Y_{jk}^{i}), \qquad \forall j, k \in \mathcal{N}, i \in \mathcal{M}, k \neq j \tag{7.13}$$

$$T_{lk} \geq T_{ij} - M(2 - Y_{kj}^{l} - X_{il}^{j}), \qquad \forall i, l \in \mathcal{M}, j, k \in \mathcal{N}, j \neq k, l \neq i \tag{7.14}$$

$$C_{max} \geq 0 \tag{7.15}$$

$$T_{lj} \geq 0, \qquad \forall j \in \mathcal{N}, l \in \mathcal{M}_d \tag{7.16}$$

$$X_{il}^{j} \in \{0,1\}, \qquad \forall j \in \mathcal{N}, i, \in \mathcal{M}, l \in \mathcal{M}_d, l \neq i \tag{7.17}$$

$$Y_{jk}^{i} \in \{0,1\}, \qquad \forall j \in \mathcal{N}, k \in \mathcal{N}_d, i \in \mathcal{M}, k \neq j \tag{7.18}$$

Objective function (7.1) is the makespan minimization, defined by the constraint set (7.2). Constraint set (7.3) ensures the start processing time for all jobs in machine dummy is zero. Constraint set (7.4) enforces each machine i to have only one predecessor in the route of machines for each job j. Constraint set (7.5) forces that each job j has only one predecessor for each machine i. Constraints sets (7.6) and (7.7) ensures that each each job k to have up to one successor in machine i and each machine l to have up to one successor in route of machines for each job j, respectively. Constraint sets (7.8) and (7.9) avoid the occurrence of one job or machine has successor or predecessor at same time. Constraint (7.10) ensure that the dummy machine 0 has one successor in route of each job j. Constraint (7.11) ensure that the dummy job 0 has one successor in each machine i. Constraint set (7.12) ensures that if job j is sequentially processed in the machines l and i, then the start time of machine i must initiate after the conclusion of this job on machine l. Constraint set (7.13) guarantees that if job k precedes job j, then the start processing time job j must be initiated after the conclusion of job k in machine i. Constraint set (7.14) guarantees the machine blocking constraints. The start time of job k on machine l must be greater than or equal to the start time of the next operation of job j that was on machine l before job k and moved to machine i. If the start time of job j on the next machine i is greater than the completion time of job j on machine l, then machine l will be blocked until operation ij is started and job j leaves machine l. Finally, constraints (7.15), (7.16), (7.17), and (7.18) define the domain of decision variables.

The second MILP (MILP2) model uses a positional-based notation for the decision variables. Each one represents a order of an operation in the solution sequence of machines and jobs. This type of notation has less number of constraints, when compared to other notation as time-index notations, in classic OSSP and do not need to define the dummy job and dummy machine anymore (NADERI *et al.*, 2011b). Next, we describe the notation and equations adopted in the developed MILP model, the parameters are the same of the first MILP model.

Indices and sets:

$j, l, e, y \in \mathcal{N}$: index for jobs {1,2,...,n}.

$i, k, r, t, z \in \mathcal{M}$: index for machines {1,2,...,m}.

Decision variables:

C_{max}: makespan.

T_{ij}: starting time of job j in machine i.

X^j_{ik}: 1 if machine i is the k^{th} machine visited by job j in its sequence, and 0 otherwise.

Y^i_{jl}: 1 if job j is the l^{th} job processed by machine i in its sequence, and 0 otherwise.

The proposed MILP formulation is presented as follows.

minimize

$$C_{max} \tag{7.19}$$

subject to

$$C_{max} \geq T_{ij} + p_{ij}, \qquad \forall i \in \mathcal{M}, j \in \mathcal{N} \tag{7.20}$$

$$\sum_{k \in M} X^j_{ik} = 1, \qquad \forall i \in \mathcal{M}, j \in \mathcal{N} \tag{7.21}$$

$$\sum_{l \in N} Y^i_{jl} = 1, \qquad \forall i \in \mathcal{M}, j \in \mathcal{N} \tag{7.22}$$

$$\sum_{i \in M} X^j_{ik} = 1, \qquad \forall k \in \mathcal{M}, j \in \mathcal{N} \tag{7.23}$$

$$\sum_{j \in N} Y^i_{jl} = 1, \qquad \forall i \in \mathcal{M}, l \in \mathcal{N} \tag{7.24}$$

$$T_{ij} \geq T_{rj} + p_{rj} - M(1 - X^j_{ik}) - M(1 - \sum_{t=1}^{k} X^j_{rt}), \qquad \forall i, r, k \in \mathcal{M}, \; j \in \mathcal{N}, \; j > 1 \tag{7.25}$$

$$T_{ij} \geq T_{ie} + p_{ie} - M(1 - Y^i_{jl}) - M(1 - \sum_{y=1}^{l} Y^i_{ey}), \qquad \forall i \in \mathcal{M}, \; j, e, l \in \mathcal{N}, \; l > 1 \tag{7.26}$$

$$T_{ie} \geq T_{zl} - M(4 - X_{ik}^{j} - X_{zk+1}^{j} - Y_{jl}^{i} - Y_{el+1}^{i}), \qquad \forall i, z, k \in \mathcal{M}, \; j, e, l \in \mathcal{N}, \; k < m - 1, \; l < n - 1 \tag{7.27}$$

$$C_{max} \geq 0 \tag{7.28}$$

$$T_{ij} \geq 0, \qquad \forall j \in \mathcal{N}, i \in \mathcal{M} \tag{7.29}$$

$$X_{ik}^{j} \in \{0, 1\}, \qquad \forall j \in \mathcal{N}, i, k \in \mathcal{M} \tag{7.30}$$

$$Y_{jl}^{i} \in \{0, 1\}, \qquad \forall j, l \in \mathcal{N}, i \in \mathcal{M} \tag{7.31}$$

Objective function (7.19) is the makespan minimization, defined by the constraint set (7.20). Constraints sets (7.3) and (7.4) guarantee that each machine i is in one position in the sequence of each job j and each job j is in one position in the sequence of each machine i, respectively. Constraints sets (7.5) and (7.6) guarantee that one machine is allocated is position k in the sequence of job j and one job is allocated is position l in the sequence of machine i, respectively. Constraint set (7.25) ensures that if job j visits machine i after machine r (not necessarily immediately) in its sequence, then the start processing time of machine i must initiate after the conclusion of this job on machine r. Constraint set (7.26) ensures that if machine i process job j after job e (not necessarily immediately) in its sequence, then the start processing time of job j must initiate after the conclusion of job e on machine i. Constraint set (7.27) guarantees the machine blocking constraints. The start time of job e on machine i must be greater than or equal to the start time of the next operation of job l that was on machine i before job e and moved to machine z. If the start time of job l on the next machine z is greater than the completion time of job l on machine i, then machine i will be blocked until operation zl is started and job l leaves machine i. Finally, constraints (7.28), (7.29), (7.30), and (7.31) define the domain of the decision variables.

7.3.3 Constraint programming model

Constraint programming is a paradigm for solving combinatorial optimization problems, especially for complex problems that cannot be easily modelled with integer linear equations (ROSSI; BEEK; WALSH, 2006). CP initially emerged in the field of artificial intelligence, but has achieved good results when applied to production scheduling problems (PINEDO, 2016).

The OSSPB problem can be modelled through constraint programming using the same sets and parameters of the MILP model and two types of decision variables: interval and sequence. Interval variables represent an operation to be processed from a job on a machine, with start, end and duration times of the operation. Sequence decision variables, on the other hand, cluster several interval variables into a set, such as the processing sequence of jobs on a machine, for example. The sequence decision variables could be used in non-overlapping constraints for scheduling problems (LABORIE, 2018). Among the main competitive solvers for constraint scheduling, the research uses IBM's CP Optimizer, which has been obtaining competitive results for production scheduling problems (LABORIE; GODARD, 2007; KELBEL; HANZÁLEK, 2011; ZARANDI; KHORSHIDIAN; SHIRAZI, 2016; GEDIK *et al.*, 2018; MENG *et al.*, 2020; YUNUSOGLU; YILDIZ, 2021; ÖZTOP *et al.*, 2021; ABREU *et al.*, 2021).

Next, we illustrate a new constraint programming (CP) model for open shop considering blocking constraints. This CP model uses interval variables to represent the operations of jobs in machines and sequence variables to represent the sequence of jobs for each machine and the sequence of machines for each job.

Indices and sets:

$i \in \mathcal{M}$: index for machines {1,2,...,m}.

$j \in \mathcal{N}$: index for jobs {1,2,...,n}.

Parameters:

p_{ij}: processing time of job j in machine i.

M: a large and positive number.

Decision variables:

x_{ij}: an interval variable for to indicate the operation of job j in machine i.

Γ_i: a sequence variable with order of x_{ij} interval variable in machine i.

Υ_j: a sequence variable with order of x_{ij} interval variable in job j.

The constraint programming model for the OSSPB is presented bellow:

minimize

$$\max_{i \in M, j \in N} \text{endOf}\left(x_{ij}\right) \tag{7.32}$$

subject to

$$\text{noOverlap}\left(\Gamma_i\right), \qquad \forall i \in \mathcal{M} \tag{7.33}$$

$$\text{noOverlap}\left(\Upsilon_j\right), \qquad \forall j \in \mathcal{N} \tag{7.34}$$

$$\text{startOfNext}\left(\Gamma_i, x_{ij}, M\right) \geq \text{startOfNext}\left(\Upsilon_j, x_{ij}, 0\right), \qquad \forall i \in \mathcal{M}, j \in \mathcal{N} \tag{7.35}$$

$$\text{interval } x_{ij}, \quad \text{size} = p_{ij}, \qquad \forall i \in \mathcal{M}, j \in \mathcal{N} \tag{7.36}$$

$$\text{sequence } \Gamma_i, \quad \text{on } \left[x_{ij}\right]_{j \in \mathcal{N}}, \qquad \forall i \in \mathcal{M} \tag{7.37}$$

$$\text{sequence } \Upsilon_j, \quad \text{on } \left[x_{ij}\right]_{i \in \mathcal{M}}, \qquad \forall j \in \mathcal{N} \tag{7.38}$$

Equation (7.32) is the makespan minimization. Constraint set (7.33) enforces machine i processes only one job at a time. Constraint set (7.34) imposes that a given job j cannot be processed simultaneously for two or more machines. Hence, a given job j is processed by only one machine at a time. Constraint set (7.35) guarantees the machine blocking constraints. The start time of the next job to be processed in the machine i after the job j must be greater than or equal to the start time of the next machine to process the job j. If this constraint is not satisfied the machine i will be blocked until the operation of the next machine to receive the job j is completed. If the job j is the last one processed on the machine i and/or the machine i is the last one to process the job j, the third argument of the startOfNext expression is returned, producing a redundant constraint and causing no impact to the logic of the CP model. Finally, constraints (7.36), (7.37), and (7.38) define the scope of decision variables. The variable x_{ij} is an interval decision variable with a duration p_{ij}, Γ_i is a variable that stores the sequence of operations of all jobs on machine i, and Υ_i is a variable that stores the sequence of machines that process the job j.

7.3.4 Comparison of models

This subsection compares the complexity of the constraints, the number of decision variables and the number of constraints for each of the exact approaches presented. We compared the main components of OSSPB with the different ways of modelling the problem with mixed-integer programming and constraint programming.

MILP1 model uses sequential notation for decision variables, but due to problem domain characteristics, such as non-overlapping constraints for machines and jobs, the model has an expressive constraint complexity. The constraint with the largest size is the one found in equation (7.15) with the worst case complexity of $\mathcal{O}\left(m^2n^2\right)$. The MILP model has altogether $mn\left[(n-1)(m-1)+\frac{3}{2}(n+m)+3\right]+m+n$ constraints and $nm\left(n+m\right)+nm+1$ decision variables, of which $nm+1$ are real and $nm\left(n+m\right)$ are binary integers (NADERI *et al.*, 2011b).

MILP2 model uses positional notation for decision variables. The constraint with the largest size is the one found in equation (7.27) with the worst case complexity of $\mathcal{O}\left(m^3n^3\right)$. The MILP model has altogether $nm\left\{4+(n-1)\left[nm\left(m-1\right)+n\right]+m\left(m-1\right)\right\}$ constraints and $nm\left(n+m\right)+nm+1$ decision variables, of which $nm+1$ are real and $nm\left(n+m\right)$ are binary integers (NADERI *et al.*, 2011b). The first model has less constraint and decision variables than the second MILP model.

The CP model uses logical constraints for the problem and applies heuristic techniques to reduce the search space for the solution. Furthermore, CP has a greater ease in finding feasible solutions, due to the exploration of the combinatorial problem domain Rossi, Beek and Walsh (2006). This results in a greater facility for modelling combinatorial problems such as production scheduling, mainly due to the possibility of using logical constraints, which reduces the amount of decision variables and model constraints.

We presented the CP model in equations (7.32) - (7.38) which uses the CP Optimizer solver notation for sequencing problems, the constraint with the largest size was presented in equation(7.35) with the worst case complexity of only $\mathcal{O}(nm)$. The CP model has a total of $n(m+1)+m$ constraints and $n(m+1)+m$ decision variables, all variables are discrete and of the interval or sequence type. Therefore, due to the CP modelling properties and forms, it has a smaller number of constraints and decision variables than MILP, which can contribute to a better performance in the OSSPST solution. However, a smaller model will not necessarily perform better than the others, therefore we will test all exact methods in the Section 7.5. The Table 21 illustrates a comparison between the key characteristics of exact approaches, for several different instance sizes. With m the number of machines and n the number of jobs.

Table 21 – Comparison of MILP and CP formulations with examples of instances sizes for OSSPB

Instances sets		MILP1			MILP2			CP	
m	n	# integer variables	# continuous variables	# constraints	# integer variables	# continuous variables	# constraints	# integer variables	# constraints
3	3	54	10	150	54	10	468	**15**	**15**
4	4	128	17	392	128	17	2752	**24**	**24**
5	5	250	26	860	250	26	11100	**35**	**35**
6	6	432	37	1668	432	37	34704	**48**	**48**
7	7	686	50	2954	686	50	90748	**63**	**63**
8	8	1024	65	4880	1024	65	208128	**80**	**80**
9	9	1458	82	7632	1458	82	431892	**99**	**99**
10	10	2000	101	11420	2000	101	828400	**120**	**120**
15	15	6750	226	54930	6750	226	10017900	**255**	**255**
20	20	16000	401	169640	16000	401	58065600	**440**	**440**
25	25	31250	626	408800	31250	626	225752500	**675**	**675**
30	30	54000	901	840660	54000	901	682779600	**960**	**960**
40	40	128000	1601	2630480	128000	1601	3898758400	**1680**	**1680**

Source: Authors.

Analyzing the Table 21, it can be observed that with the increase in the size of the problem, the decision variables and restrictions of the MILP models have a significant increase when compared to the variables and restrictions of the CP models. The CP model presented the smallest amounts of decision variables and constraints for most of the presented problem sizes. The constraints with the largest sizes for the MILPs and the CP models are the machine blocking constraints. Therefore, the results analysis section will analyze how each model handles these constraints and performs in solution quality and computational times.

7.3.5 Problem properties

In this subsection, we present six properties for the OSSPB. Firstly, we present the NP-hardness of the proposed problem. Secondly, we present a preposition about the optimal solution of instances with and without blocking. Finally, we present a lower bound for the OSSPB and two valid inequalities for exact models.

Theorem 7.3.1. *The OSSPB is NP-hard.*

Proof. Taking into consideration the set of constraints of the first MILP model (7.14), if we remove them, the constraint are relaxed and the problem becomes the classic OSSP, which is NP-Hard for $m \geq 3$ (GONZALEZ; SAHNI, 1976). □

Since Gonzalez and Sahni (1976) demonstrated that $O_3||C_{max}$ is an NP-hard problem in the ordinary sense, the $O_m|block|C_{max}$ variant also is NP-hard. In the OSSPB, there are the same constraint of OSSP with blocking of machines constraints, thus the problem can be reduced to the classical open shop.

Proposition 7.3.2. *An optimal solution for an OSSP instance with no machine blocking is a valid solution for the same instance in OSSPB.*

Proof. An optimal solution for the classical open shop with no machine blocking occurrence before all processed operations satisfy all the constraints of the mathematical models presented. Therefore, it is also a valid solution for the OSSPB. □

There is no guarantee that mixed-integer linear programming models, as well as heuristic algorithms and constraint programming, can find the global optimal solution, or even high-quality solutions. Thus, starting from a valid solution for the classic open shop might be a good start for solving OSSPB. In this sense, the determination of a lower bound, aiming at obtaining a reference for the evaluation of a given solution found, is quite relevant.

The lower bound calculation for OSSPB starts from the relaxed OSSP problem, which considers that a job can be processed on more than one machine simultaneously (PINEDO, 2016). The makespan of the relaxed problem is an estimation of the OSSP makespan and therefore is an estimate of OSSPB makespan. It is, therefore, a lower bound for the problem.

Proposition 7.3.3. *A lower bound for OSSPB is given by:*

$$LB = \max\left\{\max_{\forall i \in M}\left\{\sum_{j \in N} p_{ij}\right\}, \max_{\forall j \in N}\left\{\sum_{i \in M} p_{ij}\right\}\right\} \tag{7.39}$$

Proof. The lower bound considers there is no idle time between operations of jobs in machines. Taking into basis the well-known lower bound for the classic open shop present by Pinedo (2016), this study extend this concept to the OSSPB. The lower bound considers the relaxed OSSP without the restrictions that a job is processed on only one machine at a time and a machine processes only one job at a time. □

From the lower bound proposition, we can estimate the Big M number to be used in MILP and CP models.

Proposition 7.3.4. *A possible Big M number for the OSSPB exact models is:*

$$M = \sum_{i \in M} \sum_{j \in N} p_{ij} \tag{7.40}$$

Proof. The lower bound is the expectation of the maximum completion time of jobs and is a summation of times of jobs or machines. The Big M is a summation of processing time of jobs and machines. Since $M > LB$, there will never be an operation with a completion time greater than Big M (ABREU; TAVARES-NETO; NAGANO, 2021). Thus, this value is valid for use in the inequalities of the exact OSSPB models for the completion time of operations and blocking calculations in the CP model. $\square$

In addition to these essential properties, user cut parameters can improve solvers of mathematical models. The user cuts are a set of valid inequality constraints that are not part of the original mathematical model and do not eliminate any feasible solution to the optimization problem, i.e., they are redundant constraints to the problem (HARDIN; NEMHAUSER; SAVELSBERGH, 2008).

We can use user cuts to reduce the relaxed search space of the problem, reducing the possibility of solutions in the region with continuous values of the problem (APT, 2003). The implementation of user cuts starts from the knowledge of the problem domain, such as the lower bounds that can be used as user cuts to the objective function of the problem.

There are several applications of user cuts in scheduling problems that obtain competitive results in MILP models (HARDIN; NEMHAUSER; SAVELSBERGH, 2008; KANG; CHEN; MENG, 2019; SABERI-ALIABAD; REISI-NAFCHI; MOSLEHI, 2020). As there are also applications of user cuts in scheduling problems that use CP models Gedik *et al.* (2018), Yunusoglu and Yildiz (2021).

Equation (7.41) illustrates a user cuts added to the constraints for the improvement of exact and constraint scheduling models.

Proposition 7.3.5. *The following inequality is valid for the OSSPB:*

$$C_{max} \geq LB \tag{7.41}$$

Proof. The lower bound is an estimate of the value of the objective function of the problem through a model with relaxed constraints. Therefore, it does not eliminate any feasible solution to the problem. $\square$

Regarding user cuts for constraint-based programming models, in addition to the constraint (7.41) that can increase improvements in model solving Al-Salem *et al.* (2004), it is also possible to add a redundant constraint. The new constraint is that the number of machines used simultaneously cannot be greater than the number of total machines.

Proposition 7.3.6. *The following inequality is valid for the OSSPB with the constraint programming model:*

$$\sum_{i \in M} \sum_{j \in N} \text{pulse}\,(x_{ij}, 1) \leq m \tag{7.42}$$

Proof. The constraint indicates the number of operations performed at any one time with the function pulse and this quantity will never exceed the number of machines available because each machine processes only one job at a time. □

The extra constraint (7.42) is added to CP models to speed up the search phase of the solver by indicating the maximum amount of resources to be used at each time during the solution of the instance (GEDIK *et al.*, 2018).

7.4 Proposed solution approach

7.4.1 Initial considerations

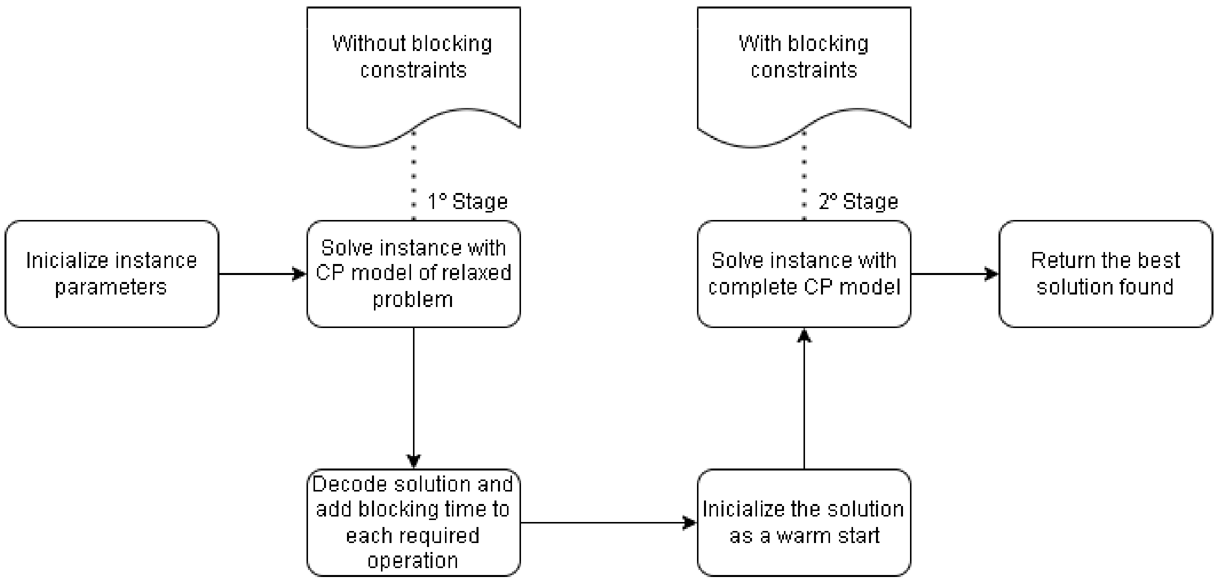
Since the OSSPB is NP-hard, a two-stage method to solve this problem is presented. We sought an algorithm with a low parameter dependency and an easy computational implementation. In the open shop scheduling problem, the operations can be sequenced in any order, increasing the number of feasible solutions to the problem and consequently the search space (NADERI *et al.*, 2010). Despite the complexity of the problem, constraint programming models have obtained excellent performances in solving the classical OSSP, as in Malapert *et al.* (2012).

However, with the addition of machine blocking constraints, the complexity of the mathematical and constraint programming models increases, as seen in the subsection 7.3.4. It makes these methods intractable for solving large-sized instances. Therefore, it is necessary to propose a flexible method to handle this type of challenging instances.

Constraint programming models have an important characteristic that it is possible to receive partial or feasible solutions with low quality for the problem (LABORIE, 2018). On the other hand, it is possible to receive the solution of the problem with relaxed constraints as a warm start to processing the search phase of the solver, which can reduce the computational time and increase the quality of solution of combinatorial optimization problems (HOJABRI *et al.*, 2018).

Therefore, this paper proposes a two-stage constraint programming (2SCP) model for solving OSSPB in competitive computational time. Figure 49 illustrates the flowchart with the execution stages of the 2SCP model.

Figure 49 – Flowchart of the proposed two-stage constraint programming approach.


Without blocking constraints
With blocking constraints
1° Stage
2° Stage
Inicialize instance parameters
Solve instance with CP model of relaxed problem
Solve instance with complete CP model
Return the best solution found
Decode solution and add blocking time to each required operation
Inicialize the solution as a warm start

Source: Authors

The first stage consists of starting the parameters of the problem instance, such as the processing time matrix p. The second stage of the algorithm consists of solving the problem with the CP model, without the machine blocking constraints (7.35). This model is the relaxed model of the OSSPB and is equivalent to the classic OSSP.

With the relaxed solution of the problem, the third phase consists of decoding the solution into a sequence representation of operations (see Naderi *et al.* (2010)). Then, we use this sequence as an encoding scheme to add the blocking constraints before processing each operation, transforming the sequence of operations into a feasible solution for the OSSPB. The encoding scheme will be described in the following subsection.

The fourth phase of the algorithm consists of converting the generated solution into the decision variables required for the CP model and then using the data as the initial solution for the search phase of the CP Optimizer constraint programming solver (LABORIE, 2018). After the warm start, it executes the CP model with the blocking constraints starting from the initial solution generated by the relaxed model. Finally, the best solution found is returned after the algorithm reaches the time limit.

The proposed exact method is considered a two-stage method because, in brief, the relaxed model of the problem is solved, and then the full problem model with the machine-blocking constraints is solved.

7.4.2 Encoding scheme

We convert the solution of relaxed the CP model in a permutation encoding the sequence of each operations machine-job and from this encoding we construct a general sequence of jobs and a sequence of machines for each job to represent a encoded scheme as a valid OSSPB solution with correct blocking constraints in before each operation.

The encoding plays a key role in the solution of the open shop with blocking constraints because the machine blocking usually incurs in a deadlock of the system. After several computational tests, we adopt an encoding based on three data structures: a vector with the sequence of operations machine-job, a vector with the sequencing of the jobs, and a matrix with the allocation of jobs per machine. Tables 4, 23, and 24 present an illustration of the proposed encoding for three jobs (J1, J2, and J3) and three machines (M1, M2, and M3) for the best solution of the example instance in Figure 48.

For the encoding scheme, we transform the solution of the relaxed CP model into a sequence of machine-job operations, such as illustrated in Table 22. This information makes it possible to obtain the sequence of jobs and machines. From the job sequence on each machine, it is possible to extract the general job sequence of the solution. The first positions of each job that appears in the job sequence on the machines will be the final positions in the job-only sequence. In the example, machine 1 processes job 2, then the next job to appear is job 1 processed on machine 2, and finally, job 3 on machine 3. So the sequence of jobs is 2, 1, and 3, present in the Table 23.

To construct the sequence of machines that process each job, we need to check the sequence of machines that appear in the processing of each job in Table 22. For example, job 1 is processed on machines 2, 3, and 1, job 2 on machines 1, 3, and 2, and job 3 on machines 3, 2, 1. With this, it is possible to build the allocation of machines for each job in Table 24.

Table 22 – Sequence of operations for the machines and jobs

Operation Index	1	2	3	4	5	6	7	8	9
Machine	1	1	1	2	2	2	3	3	3
Job	1	2	3	1	2	3	1	2	3
Sequence	**2**	**8**	**5**	**4**	**7**	**1**	**9**	**6**	**3**
Machine Sequence	1	3	2	2	3	1	3	2	1
Job Sequence	2	2	2	1	1	1	3	3	3

Source: Authors.

Considering the generated sequence, we allocate the jobs in the available machines with the basis of the allocation matrix. Given the processing times, the jobs are allocated in the machines so that two jobs cannot be processed concomitantly in different machines. Wherever possible, a given job is allocated in the machine earliest available. Subsequently,

Table 23 – Sequencing of jobs

Sequence	1^{st}	2^{nd}	3^{rd}
Job	J_2	J_1	J_3

Source: Authors.

Table 24 – Allocation of jobs per machine

job \ machine	1^{st}	2^{nd}	3^{rd}
J_1	M_2	M_3	M_1
J_2	M_1	M_3	M_2
J_3	M_3	M_2	M_1

Source: Authors.

the job starts processing with a maximum delay, aiming to avoid machine blocking. Finally, if it is not possible, we block the current machine. Figure 48 illustrates the solution represented by Tables 22, 23 and 24.

7.4.3 Complete pseudo code for 2SCP

Figure 50 illustrates the complete pseudo-code of the 2SCP algorithm. The algorithm has two main phases: the construction of an initial relaxed solution from a classic OSSP CP model and the construction of a complete solution from an OSSPB CP model with the initial solution of the first phase and a complete set of constraints. Then, when the stopping criterion of the second phase is satisfied, the best solution found (Π^{best}) is returned.

The 2SCP algorithm has just two parameters: the time limit TL_r (in seconds) of the relaxed CP model to solve OSSP and the time limit TL_c (in seconds) of the complete CP model to solve OSSPB. These parameters constitute an important trade-off between the time spent creating the partial solution to the problem and the time spent constructing a valid solution, and the search process of the CP optimizer solver. The sum of the two time limit parameters must equal the time limit defined for the 2SCP algorithm. Finally, the algorithm needs the data of the problem instance, as the processing time matrix p, the set of machines M and the set of jobs N.

The proposed algorithm also has a function $RUN\_CP\_RELAXED$ which solves the classical OSSP without the blocking constraints and generates a relaxed initial solution (x^r) of the OSSPB and a function $CREATE\_VALID\_SOLUTION$ converts a permutation encoding solution in a valid CP model with blocking solution. Finally, function $RUN\_CP\_COMPLETE$ executes a CP model for OSSPB.

Line 1 executes the CP model for OSSP without blocking constraint and with a time limit TL_r and gets a relaxed solution for OSSP. Line 2 and 3 initialize a permutation encoding Π^r and sort the operations to construct the solution of OSSP with the sequence

```
Data: $p$, $M$, $N$, $TL_r$, $TL_c$
Result: A sequence of operations $\mathit{\Pi}^{best}$
1 $x^r \leftarrow RUN\_CP\_RELAXED(p, time\_limit = TL_r)$ ;                 // 1st stage
2 $\mathit{\Pi}^r \leftarrow \{1, 2, 3, ..., mn\}$;                          // initialize solution
3 sort $\mathit{\Pi}^r_{ij} \in \mathit{\Pi}^r$ by endOf($x^r_{ij}$)  $\forall i \in M; j \in N$ in non-decreasing order;
4 $x^v \leftarrow CREATE\_VALID\_SOLUTION(\mathit{\Pi}^r, p, M, N)$ ;      // apply blocking
   constraints
5 $x^c \leftarrow RUN\_CP\_COMPLETE(p, initial\_solution = x^v, time\_limit = TL_c)$ ;
   // 2nd stage
6 $\mathit{\Pi}^{best} \leftarrow \{1, 2, 3, ..., mn\}$;                      // initialize best solution
7 sort $\mathit{\Pi}^{best}_{ij} \in \mathit{\Pi}^{best}$ by endOf($x^c_{ij}$)  $\forall i \in M; j \in N$ in non-decreasing order;
```

Figure 50 – Pseudocode of the 2SCP to solve OSSPB

of operations, respectively.

Line 4 transforms the sequence $\mathit{\Pi}^r$ in a valid solution of OSSPB with the data structures of tables 23 and 24 transformed in the decision variable x^v. Line 5 executes a CP model with blocking constraints with an initial solution x^v from a solution of OSSP and converted in a valid solution for OSSPB. The time limit of the last CP model is TL_c. Line 6 initializes a permutation encoding $\mathit{\Pi}^{best}$. Finally, sort the operations to construct the best solution found in the CP model for OSSPB with permutation encoding.

7.5 Computational experience

7.5.1 Indicators for evaluation measure

The main performance indicator adopted in the analysis of the computational experiments is relative percentage deviation (RPD) calculated as in Equation (7.43).

$$RPD = \frac{sol_{method} - LB}{LB} \times 100 \tag{7.43}$$

In this equation, sol_{method} represents the solution returned by a given method, and LB the lower bound calculated by Equation (7.39).

Aiming to determine if the difference of RPD between the two methods is statistically significant, we use the p-value as another performance indicator with a significant test. In addition, we compare the solution methods under comparison taking into account the confident interval of RPD in each benchmarking instance set.

7.5.2 Description of test instances and design

Since the problem under study was not previously reported in the revised literature, three sets of test instances are evaluated, based on the well-known benchmark problems for the classic open shop proposed by Guéret and Prins (1998), Brucker *et al.* (1997), and Taillard (1993). As stated in Ahmadian *et al.* (2021b), for OSSP, there is a gap for proposals for new challenging instance sets. Within this context, the paper also proposes a new set of large-sized instances to stress the performance of the proposed methods in real industry size applications.

For the processing times matrix, in the instances of Guéret and Prins (1998), the processing times are random values, uniformly distributed between 1 and 1000. The problem sizes are $n, m \in \{3, 4, 5, 6, 7, 8, 9, 10\}$. For each problem class, 10 instances were generated, totalizing 80. In Taillard (1993) instances, the processing times are random values, uniformly distributed between 1 and 100. The problem sizes are $n, m \in \{4, 5, 7, 10, 15, 20\}$. For each problem class, 10 instances were generated, totalizing 60. In Brucker *et al.* (1997) instances, the processing times are random values uniformly distributed between 1 and 500. The problem sizes are $n, m \in \{3, 4, 5, 6, 7, 8\}$, totalizing 60 instances. In our instances set the processing times are random values uniformly distributed between 1 and 1000. The problem sizes are $n, m \in \{25, 30, 40\}$. For each problem class, 10 instances were generated, totalizing 30. In summary, we test 222 instances for OSSPB. The size of a instances, used in next subsections, are the number of operations as number of machines plus the number of jobs ($m \times n$).

The MILP models were executed in the IBM ILOG CPLEX solver version 12.10 and the CP model were executed in the IBM ILOG CP Optimizer solver version 12.8. boths are implemented in OPL modeling language (https://www.ibm.com/br-pt/products/ilog-cplex-optimization-studio). The proposed method 2SCP was implemented in the python programming language 3.7 using DOcplex 2.10.155 libray with IBM ILOG CP Optimizer 12.10. The benchmarking methods were also implement in the python programming language 3.7. The computational experience was performed on a PC with Intel Core 2 Duo CPU 3.00 GHz and 4Gb memory, with the Windows 10 operating system.

The source codes, instance sets, results of all computational tests, and statistical analyses are available in the following link: http://repositorio.uspdigital.usp.br/handle/item/445. Another data or any questions are available upon request.

7.5.3 Results and discussion for exact models

For the exact models, we adopt a time limit of 3600 seconds in the computational tests. The tested methods are listed below.

- MILP1: the proposed mathematical programming model given by the equations (7.1)-(7.18) with sequence-based notation of decision variables.

- MILP2: the proposed mathematical programming model given by the equations (7.19)-(7.31) with position-based notation of decision variables.

- CP: the proposed constraint programming model given by the equations (7.32)-(7.38) with Auto search strategy (ABREU *et al.*, 2021).
- Mixed-Integer Linear Programming with user cuts (MILPUC): the proposed mathematical programming model given by the Equations (7.1) - (7.18) with sequence-based notation of decision variables and the user cut proposed in equation (7.41).
- Constraint Programming with User Cuts (CPUC): the proposed CP model given by the equations (7.32)-(7.38) with the user cuts proposed in equations (7.41) and (7.42) with Auto search strategy (ABREU *et al.*, 2021).

The computational tests aim at comparing the performance of the exact models in standalone form and with the use of valid inequalities, verifying if adding user cuts causes improvements in the quality of the solution. Also, it can be verified whether the proposed time limit is sufficient for the exact methods to find feasible solutions for all sets of instances, including large-sized sets such as the one proposed in the paper.

Table 25 illustrates the average RPD (ARPD) for each exact method evaluated on each set of instances. The results are grouped by instance type and size. The last four rows provide a statistical summary of the computational results: the best ARPD value (Min), the average ARPD (Average), the worst ARPD value (Max), as well as the number of unsolved instances for each exact proposed method. The results with the symbols "***" indicate that the method used did not find any feasible solution in the computational time provided for all instances in the set. The instance size (column labeled Size) is the multiplication of the machine number by the job number $m \times n$ of each instance set.

Table 15 shows that all the exact methods have not returned feasible solutions for some test instances within the considered time limit. The MILPs methods presented a greater number of unsolved instances. All the other CP-based methods presented better results, expressed by a lower average ARPD (less than 6%) and fewer unsolved instances than MILP. The MILP2 presented worse result than MILP1. Therefore, the sequence-based notation of decision variables gets best results of ARPD in OSSPB.

With respect to the exact methods with user cuts, MILPUC gets better results of ARPD in medium-sized instances (36 and 64) and (49 and 100) in Gueret and Prins and Taillard instances, respectively. When is compared to MILP1 and MILP2. However, the average ARPD for all instances of MILP1 and MILPUC is quite similar. CPUC and CP methods presents similar results of ARPD in each set of instances. Guéret and Prins (1999) demonstrates that the conventional LB of OSSP is very loose, which may explain that the improvement using LB as user cuts is not significant.

Figure 51 illustrates a boxplot of the distribution of the RPD obtained by the proposed methods on all tested instances. The dotted line is the average result for each

Table 25 – ARPD results of exact methods in all sets of instances

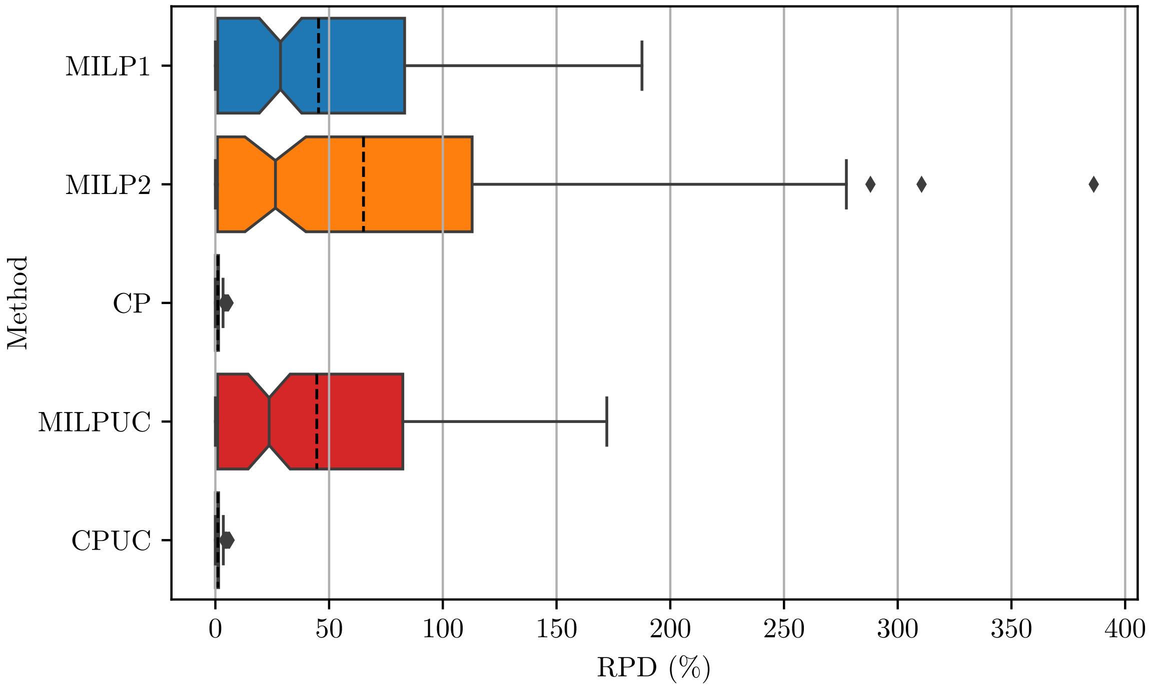
Group	Size	MILP 1	MILP 2	CP	MILPUC	CPUC
Gueret and Prins	9	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**
	16	**0.13**	**0.13**	**0.13**	**0.13**	**0.13**
	25	0.68	0.68	**0.33**	0.49	**0.33**
	36	14.92	14.70	**0.19**	4.85	**0.19**
	49	38.47	68.34	**0.00**	52.51	**0.00**
	64	75.39	134.38	**0.12**	73.18	**0.12**
	81	103.16	223.24	**0.29**	104.50	0.32
	100	139.61	255.62	**0.67**	137.31	0.86
Taillard	16	**1.81**	**1.81**	**1.81**	**1.81**	**1.81**
	25	**1.85**	4.51	**1.85**	1.95	**1.85**
	49	50.73	78.00	0.22	34.19	**0.13**
	100	91.60	117.63	0.92	77.83	**0.88**
	225	113.35	***	**0.38**	118.87	0.48
	400	125.63	***	0.68	130.25	**0.67**
Brucker	9	**1.53**	**1.53**	**1.53**	**1.53**	**1.53**
	16	**0.98**	**0.98**	**0.98**	**0.98**	**0.98**
	25	1.06	2.39	**1.04**	1.08	**1.04**
	36	15.02	38.28	**0.70**	15.81	**0.70**
	49	44.17	83.61	**1.16**	48.80	**1.16**
	64	71.64	134.00	1.36	71.39	**1.34**
Abreu et al.	625	***	***	**4.01**	***	4.08
	900	***	***	**5.31**	***	5.64
	1600	***	***	***	***	***
Min		**0.00**	**0.00**	**0.00**	**0.00**	**0.00**
Average		44.59	64.43	**1.08**	43.87	1.10
Max		139.61	255.62	**5.31**	137.31	5.64
Unsolved Instances		30	50	**10**	30	**10**

Source: Authors.

method. Overall, CP and CPUC obtained the best results, followed by MILPUC. In addition, the MILPUC has the lowest median and mean among the MILP methods. For a better comparison, Figure 52 illustrates each proposed method's ARPD, grouped by instance size.

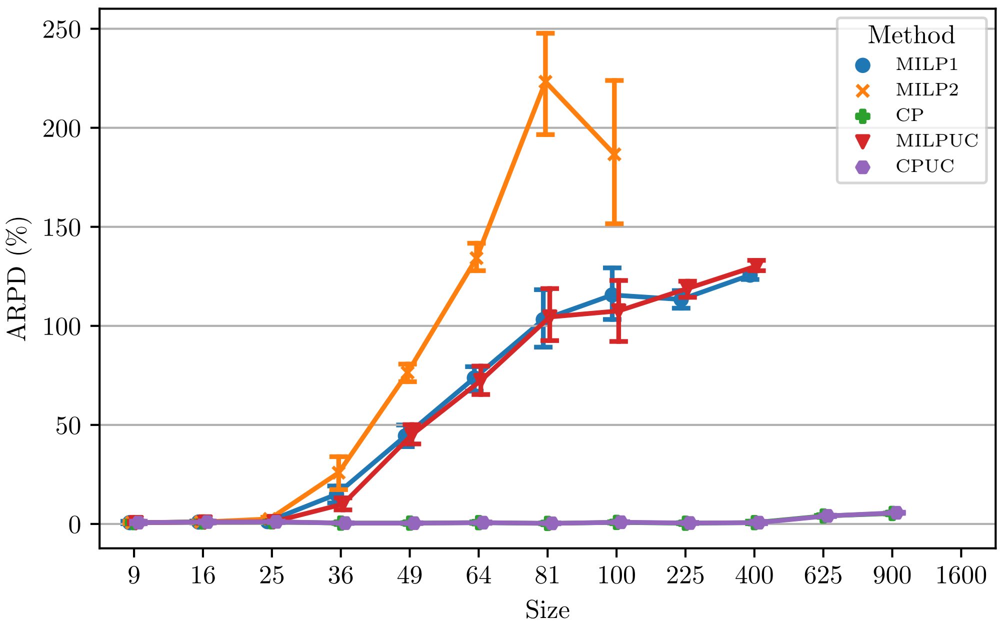
Concerning Figure 52, MILP performs well up to instance size 25, with the performance loss gradually increasing with increasing instance size, showing the difficulty of solving the OSSPB by the MILP models. The CP and CPUC methods obtained the best performance compared to the other methods in most of the instance sizes. Both methods presents similar results in each instance size (their lines overlap) and presents

Figure 51 – RPD distribution for all exact methods in all sets of instances.



Source: Authors

Figure 52 – ARPD and confidence interval ($\alpha = 0.05$) for all exact methods in each instance size.



Source: Authors

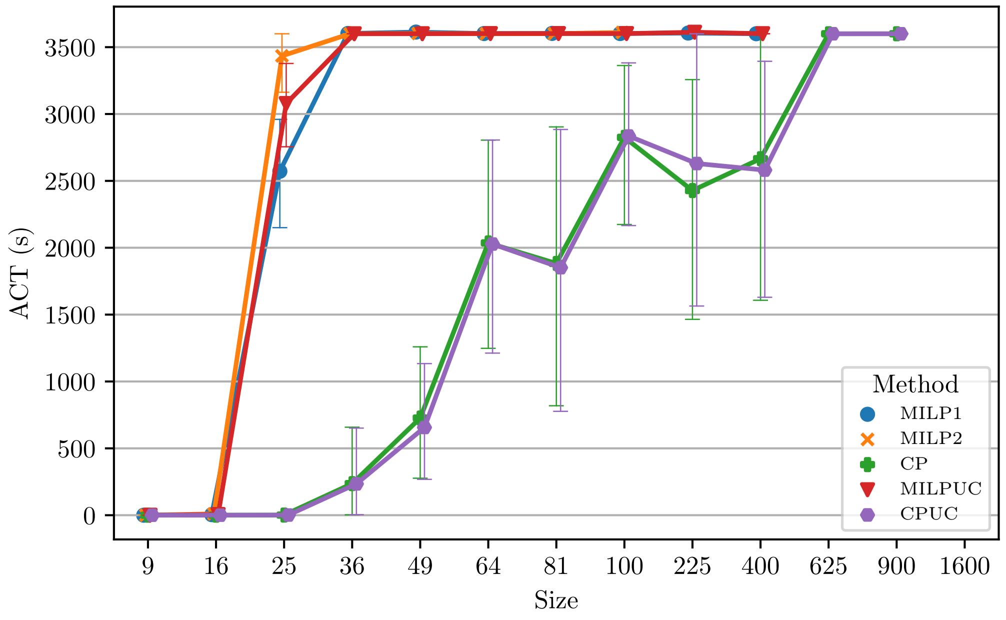
the best results between all others methods. In instance sizes larger than 100 and 400, MILP2 and (MILP1 and MILPUC) could not found valid solution with the time limit used, respectively.

MILP2 gets the worst results, with exponential growth in the difficulty of getting quality solutions. MILPUC achieves slightly better results than MILP1 on medium-sized instances (36 to 100). However, it has worse results than MILP1 on instances greater than 100.

Therefore, with the increase of complexity of the problem, considering difficult blocking constraints, solving the problem becomes very costly for instances larger than 900, with no exact method obtaining valid results with the time limit used.

Regarding computational times, Figure 53 illustrates the average computational times (ACT) obtained for each method for each instance size tested. The MILP models from instances of size 36 has significant computational times, denoting the difficulty of solving the problem by exact methods on large instances. CP and CPUC have the best computational times because it are a models with fewer decision variables and constraints, obtaining competitive times mainly for instances of sizes 36 to 400. Both get similar computational times, but CPUC gets lower ACT than CP in instances of size 40 to 81 and 400.

Figure 53 – Average computational times for all exact methods in each instance size.



Source: Authors

CP methods have the highest variability in computational times due to finding the optimal solution for some instances of a set (in competitive times) and others not (reaching the time limit). All the methods, starting from the instances of size 625, reached the 3600 seconds time limit.

Since no exact methods were able to solve all the evaluated test instances, especially for the large-sized instances, we proposed a new two-stage method to improve the performance of CP models to get better results in acceptable computational times and find valid solutions for large-sized instances. We compare the 2SCP, described in Section 7.4, with benchmarking metaheuristics from the literature of OSSP in the following subsection. The 2SCP approach adopts the CP model without user cuts (with better results between exact methods) in its two search phrases.

7.5.4 A comparison between CP and 2SCP methods

To verify the improvement of the two-stage method over the traditional CP model, we performed a comparative analysis of the two approaches. We compare 600 and 3600 seconds as the time limit of each instance for each tested method. For 2SCP, we adopt 120 or 720 seconds to TL_r for solving the relaxed model (OSSP) and 480 or 2880 seconds to TL_c for solving the complete model (OSSP) with 600 and 3600 as time limit, respectively.

Table 26 illustrates the ARPD results of the CP and 2SCP methods. The results are grouped by instance type and size. As the total execution times are similar between the two approaches, in Table 26, we perform a comparison of the time for each method to reach the best solution found (Avg. T. best column). The last four rows provide a statistical summary of the computational results similar to the summary of Table 25. The instance size (column labeled Size) is the multiplication of the machine number by the job number $m \times n$ of each instance set.

Analyzing the general results in Table 26, the CP model in both tests with different time limits did not obtain feasible solutions for all instances sets. The 2SCP model, on the other hand, obtained feasible solutions for all sets of instances, including large-sized instances. Hence, using the two-stage strategy, 2SCP obtains feasible solutions in admissible times, even within time limits of 600 seconds.

Regarding the solutions' quality for the 600 seconds time limit, the 2SCP obtained better solutions than the CP in instances sizes larger than 81 in all instance sets. For the 3600 seconds time limit, CP obtains better solutions than 2SCP in instance sizes 100 and 400, and 2SCP obtains better solutions than CP in large-sized instances up to 625. The 2SCP generates competitive solutions even for challenging instances with size larger than 400.

Regarding execution times, the 2SCP for the large-sized instances of each instance set has a shorter average time to find the best solution than the CP. Thus, besides finding similar or better solutions than the CP, the 2SCP can find these solutions faster. This result shows the efficiency of the 2SCP method concerning the traditional CP.

To illustrate the ARPD results of the two methods, Figure 54 shows the results of the RPD distribution between CP and 2SCP for the 600 and 3600 second time limits. The 2SCP method has the best median values at the tested time limits. However, the methods

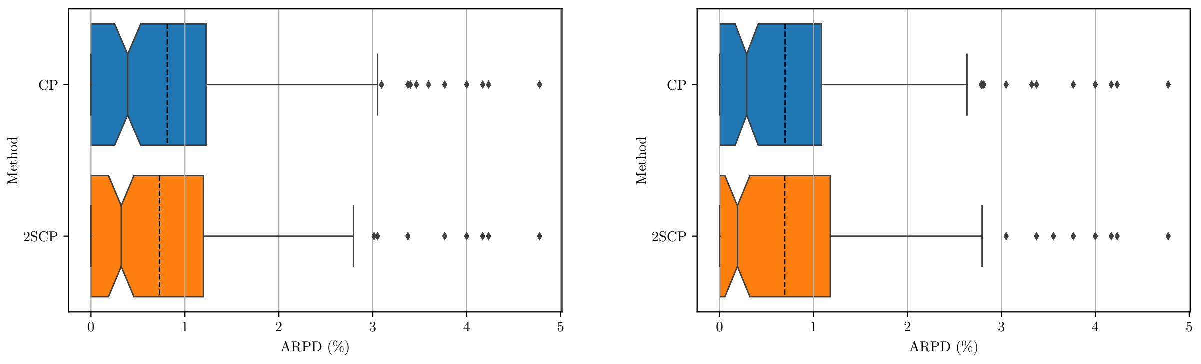
Table 26 – ARPD results and the average time to get the best solution found (Avg. T. best) of CP and 2SCP methods in all sets of instances

Group	Size	Time Limit = 600 seconds				Time Limit = 3600 seconds			
		CP	Avg T. best	2SCP	Avg T. best	CP	Avg T. best	2SCP	Avg T. best
Gueret and Prins	9	**0.00**	0.07	**0.00**	0.07	**0.00**	0.09	**0.00**	0.07
	16	**0.13**	0.09	**0.13**	0.09	**0.13**	0.12	**0.13**	0.10
	25	**0.33**	0.29	**0.33**	0.16	**0.33**	0.35	**0.33**	0.16
	36	**0.19**	0.45	**0.19**	0.22	**0.19**	0.43	**0.19**	0.21
	49	**0.00**	0.45	**0.00**	0.88	**0.00**	0.91	**0.00**	0.89
	64	**0.12**	45.10	**0.12**	80.64	**0.12**	23.21	**0.12**	80.88
	81	0.43	113.45	**0.28**	124.53	0.29	334.22	**0.25**	539.18
	100	0.94	199.14	**0.80**	191.16	**0.67**	751.82	0.80	705.08
Taillard	16	**1.81**	0.14	**1.81**	0.08	**1.81**	0.11	**1.81**	0.08
	25	**1.85**	0.69	**1.85**	0.63	**1.85**	0.67	**1.85**	0.63
	49	0.49	126.58	**0.27**	187.97	0.22	833.09	**0.07**	879.21
	100	1.48	135.51	**0.98**	234.23	0.92	1716.73	**0.88**	976.87
	225	0.57	283.54	**0.56**	326.69	0.38	923.17	**0.37**	1388.80
	400	0.88	380.66	**0.82**	375.15	**0.68**	1741.73	0.72	1631.44
Brucker	9	**1.53**	0.05	**1.53**	0.05	**1.53**	0.05	**1.53**	0.05
	16	**0.98**	0.07	**0.98**	0.07	**0.98**	0.07	**0.98**	0.07
	25	**1.04**	0.37	**1.04**	0.26	**1.04**	0.38	**1.04**	0.26
	36	**0.70**	1.17	**0.70**	1.98	**0.70**	1.20	**0.70**	1.98
	49	**1.18**	98.11	1.21	24.73	**1.16**	351.78	**1.16**	266.18
	81	2.07	198.27	**1.40**	193.54	**1.36**	1833.15	**1.36**	920.73
Abreu et al.	625	73.49	522.29	**4.11**	470.91	4.01	2071.30	**3.83**	2226.71
	900	***	***	**6.05**	472.45	5.31	1985.72	**5.01**	2535.76
	1600	***	***	**221.47**	439.33	***	***	**197.16**	2509.20
Min		0.00	0.05	0.00	0.05	0.00	0.05	0.00	0.05
Average		4.30	100.31	10.72	135.90	1.08	571.38	9.58	637.59
Max		73.49	522.29	221.47	472.45	5.31	2071.30	197.16	2535.76
Unsolved Instances		20	-	**0**	-	10	-	**0**	-

Source: Authors.

have similar overall mean and outliers to the CP method.

Figure 54 – RPD distribution for CP and 2SCP comparison in all sets of instances



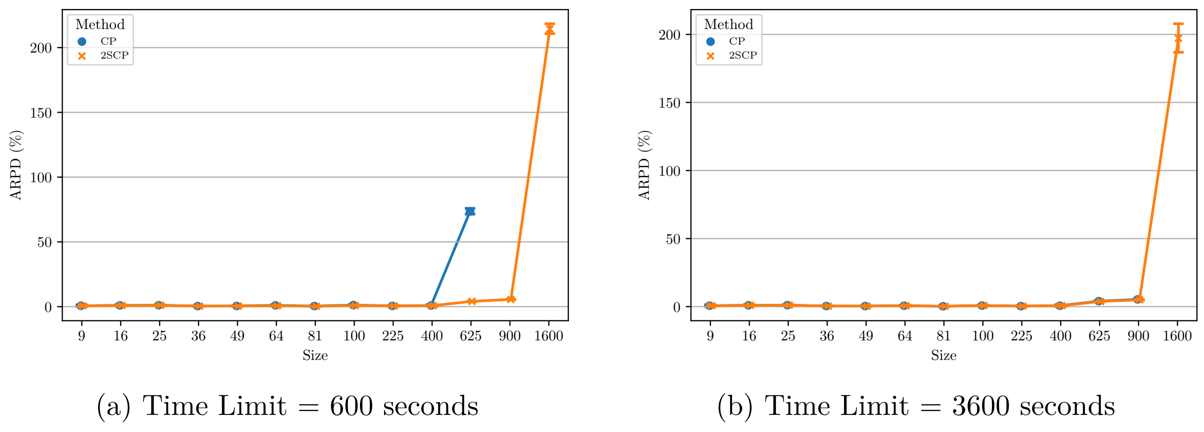
(a) Time Limit = 600 seconds (b) Time Limit = 3600 seconds

Source: Authors

Furthermore, as the time limit increases, the medians of RPD become more similar.

However, Table 26 indicates that at shorter time limit(600 seconds), 2SCP is able to obtain better solutions on most sets of instances. Therefore, adding two stages in CP makes it possible to find better solutions in a shorter time than the traditional CP approach. Regarding the results per instance size, Figure 55 illustrates the results of the ARPD between CP and 2SCP for the tested time limits and each instance size.

Figure 55 – ARPD and confidence interval ($\alpha = 0.05$) for CP and 2SCP comparison in each instance size.



(a) Time Limit = 600 seconds (b) Time Limit = 3600 seconds

Source: Authors

Analyzing Figure 55, with the time limit of 600 seconds, the CP model cannot obtain valid solutions for instance sizes of 900 and 1000, and for instance sizes 625, it obtains a worse ARPD than 2SCP. In addition, for the tests with a time limit of 3600 seconds, the CP model can obtain solutions for size 900, but only 2SCP can obtain feasible solutions for size 1000. Thus, the 2SCP strategy is competitive in large-sized instances.

7.5.5 Results and discussion for benchmarking metaheuristics and the proposed two-stage method

For this testes we compare the proposed method 2SCP with benchmarking metaheuristics from the literature of OSSP and adapted for OSSPB by us and a adaptation of MILP in the same procedure of 2SCP as a two-stage MILP model (2SMILP) for comparison purposes with the new 2SCP. We adopt 600 seconds as a time limit of each instance for each tested metaheuristic. We adopted the same stopping criteria for all algorithms for a fair comparison. With this time limit, we can test if the approximation methods obtain quality solutions with a competitive computational times. Each of the metaheuristics, due to stochastic behavior, tested were executed five times, the best and average value found for each instance set is reported in the results.

For 2SCP and 2SMILP we adopt 120 seconds to TL_r for solving the relaxed model (OSSP) and 480 seconds to TL_c for solving the complete model (OSSP). The solution time for the classical OSSP problem using the CP model is fast, even obtaining optimal solutions with less than 1 minute for large instances, as in the study of CP models for OSSP of Malapert *et al.* (2012). In 2SMILP the relaxed model got competitive results in OSSP (NADERI *et al.*, 2011b). Thus most of the execution time is allocated to solving the OSSPB problem. Hence, the total execution time of 2SCP and 2SMILP is 600 seconds to compare with the benchmarking metaheuristics.

For the computational testing of the approximation approaches, we test several competitive metaheuristics from the OSSP, adapted for the variant considered in the research, for comparison with 2SCP and 2SMILP. In summary, we test 7 approximate approaches for OSSPB.

- Electromagnetic Heuristic (EH): a complete metaheuristic to solve OSSPB with total completion time minimization proposed by Naderi, Najafi and Yazdani (2012) and adapted to makespan minimization by us.
- Extended Genetic Algorithm (EGA): a algorithm to solve classic OSSP proposed by Hosseinabadi *et al.* (2018) and adapted for OSSPB by us.
- Genetic Algorithm with restart procedure and direct decoding mechanism of solution (GA (D)): a competitive algorithm to solve OSSP with setup times proposed by Abreu *et al.* (2020) and adapted to OSSPB by us.
- Hybrid Genetic Algorithm (HGA): a improved genetic algorithm for OSSP proposed by Ahmadizar and Farahani (2012) and adapted to OSSPB in this study.
- A self-tuning variable neighborhood search algorithm (VNS): a metaheuristic proposed by Mejía and Yuraszeck (2020) for OSSPST and adapted to OSSPB by us.
- Two-Stage Constraint Programming method (2SCP): the proposed two-stage method to solve OSSPB with a hybridization of solving classic and with blocking open shop problems.
- Two-Stage Mixed-Integer Linear Programming method (2MILP): the same procedure and phases of 2SCP in flowchart 49 with the proposed MILP1 model with ($1^{\underline{o}}$ phase) and without ($2^{\underline{o}}$ phase) blocking constraint (7.14).

For the EH, EGA, GA (D), HGA and VNS the same parameters used by Hosseinabadi *et al.* (2018), Naderi *et al.* (2010), Abreu *et al.* (2020), Ahmadizar and Farahani (2012) and Mejía and Yuraszeck (2020), respectively, were considered in the tests. In EH we use the parameters: population size 10, initial temperature 20, CT 2, and FN 20. In EGA we use the parameters: population size 92, crossover ratio 0.8, and mutation ratio 0.2. In GA (D) we use the parameters: population size 100, mutation ratio 0.05, restart ratio 0.5, and initial temperature 100. In HGA we use the parameters: population size 200, crossover rate 0.9, mutation rate 0.2, local optimization heuristic rate 0.2, maximum number of iterations of the local optimization heuristic 200 and q 0.9. Finally, in VNS we use the parameters: a 50 and decoding scheme m-AS. For a fair comparison with metaheuristics, the two-stage methods 2SCP and 2SMILP use one thread only due to metaheuristics not using parallelization.

Table 27 shows the best and average results of the ARPD of approximation methods in each set of instances. Analyzing Table 27, it can observe that feasible solutions are found for all the methods under comparison for all the test instances. The instance size (column labeled Size) is the multiplication of the machine number by the job number $m \times n$ of each instance set.

Table 27 – ARPD results of approximate methods in all sets of instances

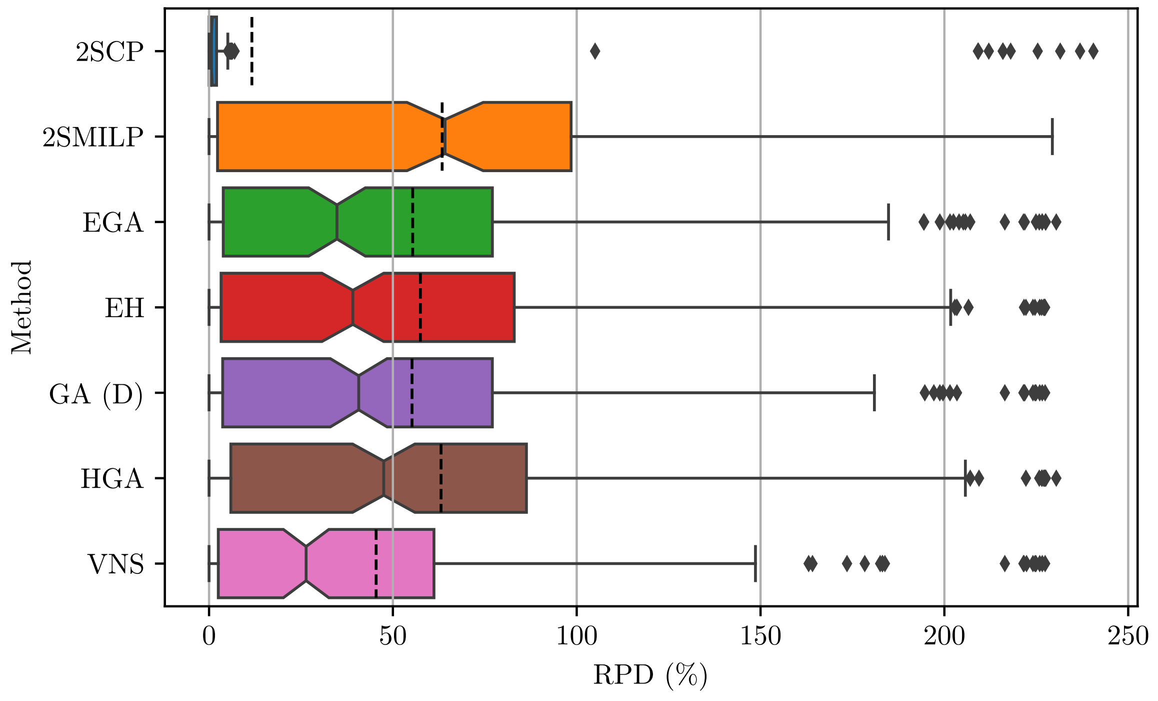
Group	Size	2SCP	2SMILP	EGA		EH		GA (D)		HGA		VNS	
				Best	Avg	Best	Avg	Best	Avg	Best	Avg	Best	Avg
Gueret and Prins	9	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**
	16	**0.13**	**0.13**	0.19	0.42	0.17	0.42	0.17	0.17	0.43	0.88	0.17	1.97
	25	**0.33**	1.09	1.84	3.33	1.51	1.91	1.52	2.24	3.98	5.97	1.26	3.50
	36	**0.19**	32.06	4.04	12.27	3.16	4.72	4.23	7.15	11.31	24.70	2.82	4.68
	49	**0.00**	65.40	19.23	31.66	18.93	26.33	30.08	35.89	44.14	51.16	9.72	11.91
	64	**0.12**	82.74	42.81	54.09	43.43	52.47	47.23	57.91	66.41	72.95	30.17	32.50
	81	**0.32**	104.82	60.40	73.68	63.95	73.05	68.51	78.03	80.79	85.12	46.33	49.12
	100	**0.94**	118.15	78.57	88.30	85.13	89.74	82.50	91.72	87.60	90.60	65.47	69.15
Taillard	16	**1.81**	**1.81**	3.52	5.34	2.67	3.11	2.52	2.67	3.52	4.87	2.52	4.67
	25	**1.85**	3.76	14.46	21.19	11.60	17.81	11.97	17.44	16.91	18.66	4.46	6.63
	49	**0.58**	61.65	39.05	45.56	41.89	45.24	43.57	47.14	45.67	48.16	30.17	32.78
	100	**1.54**	88.64	58.34	63.66	62.79	67.09	60.08	66.54	64.57	66.86	48.41	51.97
	225	**1.01**	100.86	79.85	85.94	89.95	92.88	77.56	84.36	89.95	92.61	65.68	69.51
	400	**1.28**	107.20	101.23	108.32	108.80	112.74	92.04	97.98	108.80	111.72	80.43	84.75
Brucker	9	**1.53**	**1.53**	1.59	1.59	1.59	1.59	1.59	1.59	1.59	3.37	1.59	3.88
	16	**0.98**	**0.98**	2.41	4.76	1.93	2.11	1.93	1.99	2.47	3.84	1.93	4.08
	25	**1.04**	4.99	11.49	19.75	11.80	14.60	12.19	17.31	17.05	19.13	4.71	6.67
	36	**0.70**	44.58	26.24	36.91	29.46	35.86	32.37	37.68	38.48	40.65	16.22	18.54
	49	**1.25**	79.84	46.20	53.95	49.32	55.93	51.82	59.14	58.29	60.56	35.72	38.59
	64	**1.87**	85.31	59.65	68.65	63.72	70.04	62.53	70.08	73.08	75.69	46.72	49.82
Abreu et al.	625	**14.76**	182.33	175.16	181.70	180.81	185.95	151.27	161.30	181.89	186.11	135.42	138.95
	900	**6.05**	206.70	199.33	205.10	200.41	205.45	188.99	192.13	203.62	208.17	170.84	176.78
	1600	**221.47**	***	224.93	230.27	225.38	229.20	223.51	224.36	226.80	231.38	223.56	239.83
Min		**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**
Average		**11.29**	62.48	54.37	60.71	56.45	60.36	54.27	58.91	62.06	65.36	44.53	47.84
Max		221.47	206.70	224.93	230.27	225.38	229.20	223.51	224.36	226.80	231.38	223.56	239.83
Unsolved Instances		**0**	10	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**

Source: Authors.

In the results of Table 27, the 2SCP, as a two-stage method, could find valid solutions for all tested instances, including the large-sized instances. The classic CP model could not obtain valid solution for instances of size larger than 900 with the same time limit of 600 seconds used in this test (results of Tables 26). This may indicate that adopting a two-stage solution procedure in CP model can promote improvements in the CP Optimizer execution. Finally, the 2SMILP could not obtain valid solution for instances of size larger than 900.

Furthermore, the 2SCP returned the best values for Min, Average, and Max performance indicators. Comparing the best results with the average results of the 2SCP, both have similar values, which might indicate that the 2SCP has low variability of results. Also, about Table 27, the bold values illustrate the best ARPD results for each instance set. The 2SCP method was the one that obtained the best results for each of the tested instance sizes. Figure 56 illustrates a boxplot of the distribution of RPD obtained by the approximate methods on all tested instances. The dotted line illustrates the average result for each method.

Figure 56 – RPD distribution for all approximation methods in all sets of instances.

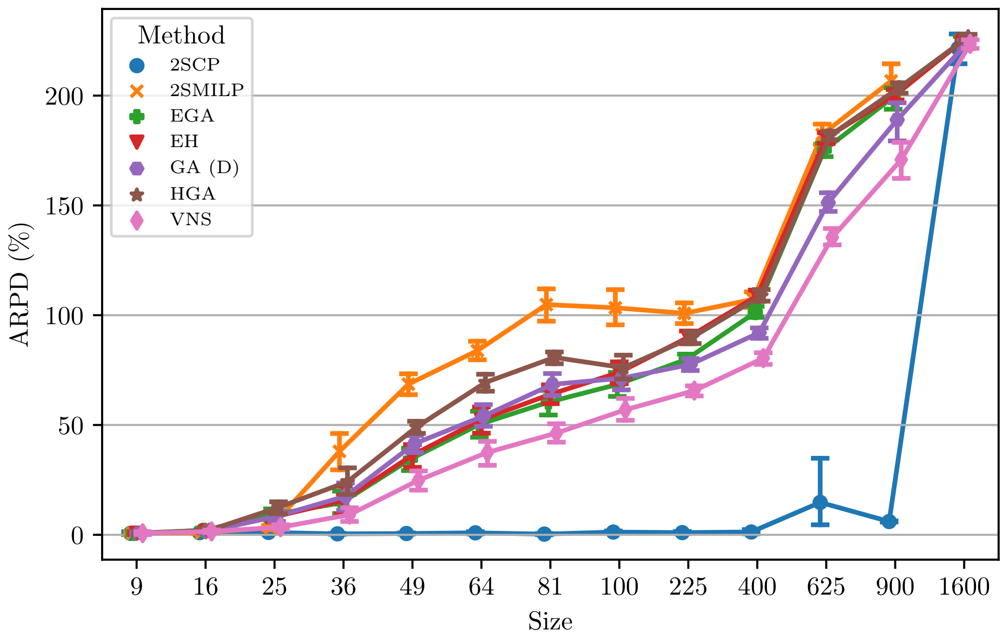


Source: Authors

In general, 2SCP obtained the best results, followed by VNS and GA (D). Comparing the proposed method and the tested metaheuristics and 2SMILP, the proposed 2SCP has the lowest mean, median, and outliers values, thus the two-stage method with the CP model is more competitive than with the MILP model. For a better comparison, Figure 57 illustrates the ARPD of each of the approximate methods tested, grouped by instance size.

From Figure 57, the metaheuristics have good performance up to instance size 16; for instances with larger sizes, the methods obtained the worse results than 2SCP. The EGA, GA (D) and VNS methods had similar results on most instance sizes, with VNS having better results on large-sized test instances of sizes 36 to 900. These methods had ARPD results larger than 50% on the larger instances, which shows that the problem considering blocking constraints has a costly resolution for these metaheuristics.

Figure 57 – ARPD and confidence interval ($\alpha = 0.05$) for all approximation methods in each instance size.



Source: Authors

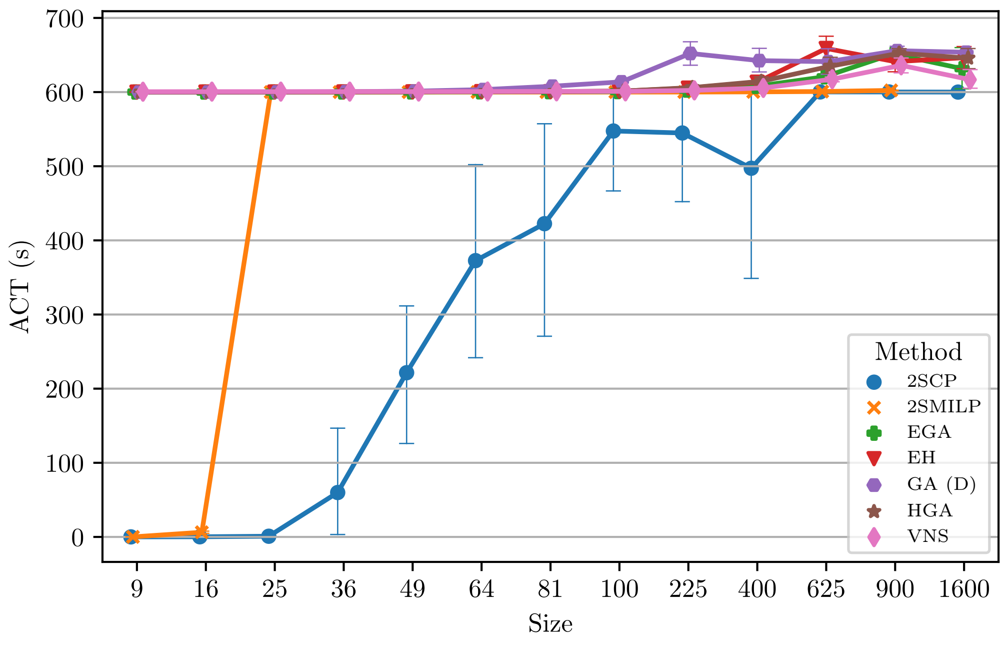
The 2SMILP method got competitive ARPD results in instances sizes up to 25 but is outperformed by other methods in instances sizes between 36 to 225. For instance sizes greater than 225, 2SMILP got similar performance as EGA and EH metaheuristics. The 2SCP method obtained the best performances about the others in most of the instance sizes, with a significant difference mainly for the intermediate instances sizes between 25 to 900, with the confidence interval not crossing with most of the other methods. However, in the challenging instances of size 1600 the methods got similar results with 2SCP exponentially decreasing its performance, but still has a better ARPD than the metaheuristics and 2SMILP.

The growth of the ARPD from size 900 to 1600 indicates an increase in the difficulty of solving large-sized instances. Still, 2SCP obtained viable solutions with relative quality when compared to other metaheuristics and 2SMILP, while the standard CP did not obtain any viable solution in the time limit of 3600 seconds for all instances of the set. These results indicate the competitive performance of the 2SCP concerning the standard CP with adopting the two-stage solution method.

Regarding computational times, Figure 58 illustrates the average computational times obtained for each method for each instance size tested. We can see when 2SCP reaches the available time limit of 600 seconds.

Analyzing the Figure 58, 2SCP manages to obtain the optimal solution of sets of some small and medium-size instances before reaching the time limit. Starting at instances

Figure 58 – Average computational times for all approximation methods in each instance size.



Source: Authors

size 625, the 2SCP can reach the time limit (600 seconds) for all instances and the 2SMILP can reach the time limit in instances size larger than 16. In addition, some metaheuristics go a bit past 600 seconds on large instances due to the high computational cost of the last iteration, eventually reaching the time limit. 2SCP algorithm have the highest variability in computational times due to finding the optimal solution for some instances of a set (in competitive times) and others not (reaching the time limit).

It is essential to check whether the differences in ARPD values are statistically significant between the approximate methods tested to validate the results. Since the data are not normally distributed and do not have similar variances, non-parametric tests are recommended (LATORRE *et al.*, 2020). We perform a Kruskal-Wallis test for non-parametric analysis of variance (MONTGOMERY, 2017), the p-value is very close to zero.

Table 28 illustrates the non-parametric Mann-Whitney rank test to see if the differences in ARPD means between the proposed 2SCP method and the other benchmarking algorithms are significant. Table 28 shows the difference of ARPD of 2SCP compared with other methods (Average diff. (%)). 2SCP obtained significantly better results when compared peer-to-peer with all other benchmarking methods.

Therefore, analyzing the results present in Table 27 and Figures 57 and 58. 2SMILP, EGA, EH, GA (D), HGA and VNS methods obtained good solution in small-sized instances. However, 2SCP outperformed the tested benchmarking methods 2SMILP, EGA, EH, GA

Table 28 – Mann-Whitney rank test between 2SCP and all other benchmarking methods.

2SCP vs.	Average diff. (%)	Statistic U	p-value
2SMILP	-61.65	9799	**0.0000**
EGA	-43.74	8392	**0.0000**
EH	-45.83	8991	**0.0000**
GA (D)	-43.55	8867.5	**0.0000**
HGA	-51.45	74478	**0.0000**
VNS	-33.79	9516	**0.0000**

Source: Authors.

(D), HGA and VNS concerning solution quality in the most set of instances. Therefore, 2SCP is so far considered a competitive solution procedure for OSSPB, with a good trade-off between solution quality and computational cost.

7.6 Final remarks

In this paper, a variant for the open shop scheduling problem considering machine blocking (zero buffer constraints) is presented. The objective function is minimizing the total time to complete the schedule (makespan). A two-stage constraint programming algorithm is presented to solve this challenging problem.

The expressive results of the proposed approach in the well-known classical open shop benchmarks proposed by Guéret and Prins (1998), Taillard (1993) and Brucker *et al.* (1997) and the new set of challenging large-sized instances proposed by us, point to its efficacy and efficiency for solving open shop scheduling problems with machine blocking constraints.

Computational experiments were carried out in order to evaluate the performance of the proposed two-stage method. To the best of the authors' knowledge, there is no other work that has presented an open shop scheduling environment with machine blocking for the makespan minimization.

Although the computational effort used by the proposed two-stage method was smaller than the computational time required and ARPD for MILP models for the all test instances, for the medium-sized and large-sized problems the proposed algorithm outperformed the MILP models and the benchmarking metaheuristics tested in ARPD indicator. In addition, 2SCP could solve problems with large-sized instances that classic CP model could not solve. Thus, the proposed approach can be used to solve real-world problems.

As extensions of this work, the proposed an local search operator for increasing the efficiency of 2SCP in solving large-sized problems is recommended. Other open shop variants could be investigated, considering different performance measures, such as total completion time as well as total tardiness minimization. The development of hybrid algorithms, such as matheuristics or constraint programming approaches with better valid inequalities, is another research opportunity. Finally, it is possible to easily adapt the exact method approach developed in the paper to solve other emerging production scheduling problems, including practical cases in real-word factories.

8 A NEW VARIABLE NEIGHBORHOOD SEARCH WITH CONSTRAINT PROGRAMMING SEARCH STRATEGY FOR OPEN SHOP SCHEDULING PROBLEM WITH OPERATION REPETITIONS

8.1 Introduction

Shop scheduling problems are widely studied optimization problems because of their several industrial applications. In the last decades, variants of the flow shop scheduling problem and job shop scheduling problem have received a lot of attention by the researchers of production scheduling area. Nevertheless, despite its theoretical and practical importance, open shop scheduling problem (OSSP) has received much less attention. Gonzalez and Sahni (1976) firstly proposed the OSSP. The classical open shop can be defined as follows: let n jobs that must be processed in m machines, in which each job operation presents an associated processing time. The problem concerns in the determination of a job sequence for the optimization of a given objective function, usually the makespan or the total tardiness.

Unlike flow shop and job shop scheduling problems, in the OSSP there are no predefined routes for the jobs in the machines, which can process all jobs. Thus, the number of viable solutions is significantly higher than the classical production scheduling problems, such as flow shop and job shop. One can observe that the processing of the jobs in the machines occurs in distinct moments, i.e. a given job cannot be processed concomitantly for more than one machine. In the non-preemptive case of the OSSP, the processing of a given job cannot be stopped until the end of the task. OSSP is suitable for various industrial applications, such as: plastic injections, chemical processes, the oil industry, food production and pharmaceutical production. In the service sector, this problem can be modelled for scheduling medical services, museum visits and telecommunications (GONZALEZ; SAHNI, 1976; LIN; LEE; PAN, 2008; VINCENT; LIN; CHOU, 2010; NADERI; NAJAFI; YAZDANI, 2012; ABREU *et al.*, 2020; BAI *et al.*, 2017).

Taking into consideration one machine (m=1), the open shop can be reduced to a single machine problem. For the case of two machines (m=2), there are polynomial algorithms with optimality proof (GONZALEZ; SAHNI, 1976). For problems with three or more machines ($m \geq 3$), the OSSP is NP-hard (GAREY; JOHNSON, 2012). Although have been proposed some branch-and-bound algorithms and mathematical programming approaches for open shop (BRUCKER *et al.*, 1997; GUÉRET; PRINS, 1999; GUÉRET; JUSSIEN; PRINS, 2000; OZOLINS, 2019), exact methods are quite limited for solving large-sized problems.

Therefore, the proposition of heuristic or metaheuristics algorithms is of great importance in order to obtain solutions to NP-Hard optimization problems. The development of metaheuristics for scheduling problems can contribute to obtaining high quality solutions with competitive computational times, especially in large-sized problems present in several industrial applications. There are many recent metaheuristics algorithms to solve optimization problems in production scheduling (HAN *et al.*, 2016; HAN *et al.*, 2017; GONG; HAN; SUN, 2018).

This article aims at investigating a new variant of the open shop problem considering the processing of more than one operation per job by a same machine, the open shop scheduling problem with repetitions (OSSPR). The problem is to process a set of jobs, with a list of operations, in a set of machines. Each machine processes one operation at a time, and each machine may process multiple operations of the same job. The objective is to find a processing sequence of operations that minimizes the total scheduling duration (makespan).

According to the literature review, the OSSPR has not been reported yet (AHMADIAN *et al.*, 2021a). The authors propose an innovative variable neighborhood search (VNS) using variable search strategy with constraint programming (CP) to provide good solutions with admissible computational effort. The new proposal outperformed seven heuristics and a competitive meta-heuristic adapted from the classic variant of the open shop.

The main theoretical and experimental contributions are threefold. First, an innovative constraint programming approach is developed with an efficient structure to model the most complex constraints of the problem. In addition, the proposed VNS uses several search strategies from the CP model as local search mechanisms. Finally, the VNS hybridized with CP presented better results than other methods, such as integer programming, heuristics, and metaheuristics in three sets of randomly generated sets instances adapted from the literature.

The remainder of this article is organized as follows: Section 2 presents the literature review as well as the proposed innovation, Section 3 describes the scheduling problem treated in this article, Section 4 describes the proposed solution approach, Section 5 presents results discussion from computational experiments; finally, in Section 6 there are some conclusions and suggestions for future works.

8.2 Related approaches and proposed innovation

In a flow shop scheduling problem, all the jobs must be processed with a fixed route, while in a job shop scheduling problem each job presents its own processing route in the available machines. Conway, Maxwell and Miller (2003) presented a problem named randomly routed job shop in which a given job is a set of operations strictly ordered. In other words, each job presents its technological precedence. Based on this concept, in this production environment, there is not a predefined sequence of machines for the several operations of a given job; however, these operations must be processed in a predefined technological order.

On the other hand, in the available literature has been considered that the difference between the open shop and flow shop or job shop is that in the open shop, there is no precedence relations between the operations of the same job and a given operation must be processed for a determined machine (GONZALEZ; SAHNI, 1976; SHA; HSU, 2008; ANAND; PANNEERSELVAM, 2016). Thereby, the open shop concept refers to the fact of the operations of a given job are technologically independent. In addition, there is an indexation of the operations to determined machines, as well as a job cannot present two or more operations processed in the same machine.

In the view of the complexity of the open shop scheduling problems, approximate algorithms have been proposed for their solution. Liaw (1999) presented a tabu search that uses blocks of operations on a critical path as the neighborhood structure. In addition, an efficient procedure is proposed for evaluating a neighborhood. Extensive computational tests point to the efficiency of the proposed approach. Prins (2000) proposed a Genetic Algorithm (GA) with two special features: a population with individuals with different makespan values and a procedure for the reordering of the generated chromosomes. This simple as fast algorithm could reach high quality solutions in comparison with the best known heuristics and metaheuristics. Sha and Hsu (2008) presented a new Particle Swarm Optimization (PSO) algorithm with an innovative encoding for the particles and a particle movement based on an insertion operator. The computational results presented several new best known solutions for the unsolved problems. Naderi *et al.* (2010) proposed four constructive heuristics taking into account problem properties, which have outperformed all the other constructive heuristics in the reported literature. One of such heuristics adopts a local search procedure, aiming to improve the initial solution found. More recently, Hosseinabadi *et al.* (2018) proposed an extended genetic algorithm and the hybrid genetic operations produced high-quality results in a set of small-sized random generated instances, as well as in the well-known Taillard benchmark instances (TAILLARD, 1993).

In the last few years, new variants of the open shop have been proposed. Roshanaei, Esfehani and Zandieh (2010) presented an open shop scheduling problem with sequence-dependent setup times. Bai and Tang (2013) proposed an open shop problem considering makespan minimization and release dates. Naderi and Zandieh (2014) studied a no-wait open shop problem, in which there are no intermediate buffers between the machines. Also, Bai, Zhang and Zhang (2016) presented a flexible open shop, in which the scheduling considers concomitantly the processing orders and the route of the jobs through production stages. Mosheiov *et al.* (2018) and Sheikhalishahi *et al.* (2019) addressed open shop environments taking into account maintenance issues. Aghighi *et al.* (2021) presented a open shop environment with reverse job flows.

Timkovsky (2004) addressed a class of shop scheduling problems, denominated cycle shop scheduling problems, in which some jobs can be reprocessed on some machines with a given number of repetitions. This situation occurs in the manufacturing of microchips in a VLSI technology environment. In the open shop scheduling problem with repetitions (OSSPR), it study an open shop scheduling problem in which the jobs can be processed in any machine more than once (operation by operation). Thereby, all the jobs can be scheduled in an unconstrained way, increasing substantially the number of feasible solutions, in comparison with the classical OSSP.

Among many variants of shop scheduling problems, several recent studies have addressed the flexible open shop (WITKOWSKI; ANTCZAK; ANTCZAK, 2012; AZADEH *et al.*, 2014; BAI; ZHANG; ZHANG, 2016). The flexible open shop problem considers a set of parallel machines in each production stage, where each available machine can process a given job. On the other hand, in the problem under study, it considers a single machine in each production stage, where each machine can process a given job more than once, depending on the number of operations required for the job.

The table 29 illustrates the main contributions of the literature to the OSSP considering new variants similar to operations repetitions. The authors, year of publication, characteristics of the problem such as the type of setup, solution methods and main research contributions are illustrated.

Table 29 – Summary of the main contributions from the OSSP literature considering new variants similar to operations repetitions.

Author	**Problem characteristics**	**Solution method**	**Contribution**
Roshanaei, Esfehani and Zandieh (2010)	Setup times	SA	Proposed a new multi-neighborhood search to improve SA and got competitive results in Taillard instances.
Bai and Tang (2013)	Release dates	Heuristic dense sheduling-based	New online heuristic and a proof for asymptotically optimal lower bound.
Naderi and Zandieh (2014)	Blocking	EH	EH outperforms MILP in small and large-sized instances.
Bai, Zhang and Zhang (2016)	Flexible OSSP	Diferential evolution	The algorithm obtain high-quality solution in moderate-scale problems.
Mosheiov *et al.* (2018)	Maintenance actives	Aproximation algorithm	Computational experimets with two machine OSSP with maintenance start windows got competitive reults.
Sheikhalishahi *et al.* (2019)	Human error and maintenance actives	NSGA-II	A real case study is solved by the NSGA-II multiobjective metaheuristic. The method found near optimal results.
Aghighi *et al.* (2021)	Reverse flows	MILP and vibration damping optimization	The proposed method outperforms al other tested methods in large-scale problems.

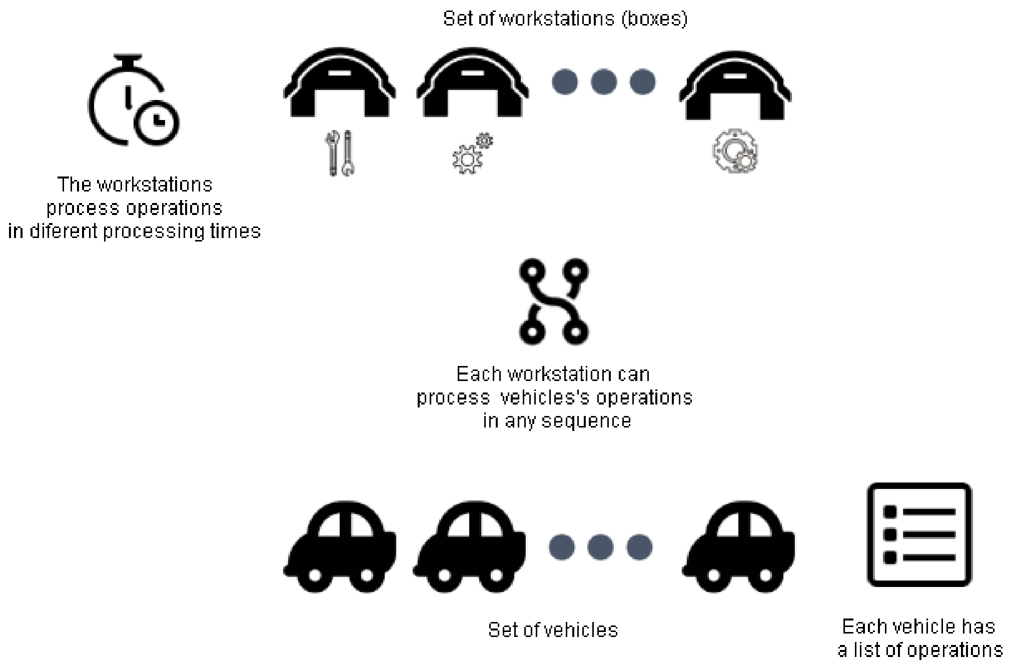
Source: Authors.

An example of a real-world application of the proposed variant can be described as follows: it is considered a large automotive garage for vehicle maintenance and repair. Each workstation (box) has a mechanic with a set of tools. Consequently, each workstation can receive any type of vehicle and perform all of the required operations. In the view of the heterogeneity of the workstations, expressed in terms of the ability of each mechanic as well as features of the available tools, each workstation can perform the jobs with different processing times.

The set of vehicles are the jobs to be processed in a set of workstations (boxes). These workstations are machines and process a set of maintenance activities in vehicles. These activities are a set of operations of vehicles, and each workstation has mechanics with different abilities. Therefore, the vehicle's operations have different processing times in each workstation.

Figure 59 shows an example of the maintenance system as an example of the proposed problem. The set of vehicles are the jobs to be processed in a set of workstations (boxes). These workstations are machines and process a set of maintenance activities in vehicles. These activities are a set of operations of vehicles, and each workstation has mechanics with different abilities. Thus, the vehicle's operations have different processing times in each workstation.

Figure 59 – A practical example of OSSPRM


Set of workstations (boxes)
The workstations process operations in diferent processing times
Each workstation can process vehicles's operations in any sequence
Set of vehicles
Each vehicle has a list of operations

Source: Authors

This characteristic addresses a new open shop environment in which a given job can be processed, for different operations, more than one time for the same machine. All operations can be processed on all machines, as well as a given machine can perform all operations for each job. In a recent literature survey for the open shop scheduling problem (ANAND; PANNEERSELVAM, 2016; ADAK; AKAN; BULKAN, 2020), there is no mention of the variant proposed in this article.

Table 30 – Processing times per operation for each job

$p_{ij}^k \setminus J$	J_1		J_2	
$O_j \setminus M$	M_1	M_2	M_1	M_2
k_1	2	2	5	3
k_2	1	3	1	2
k_3	3	2	-	-

Source: Authors.

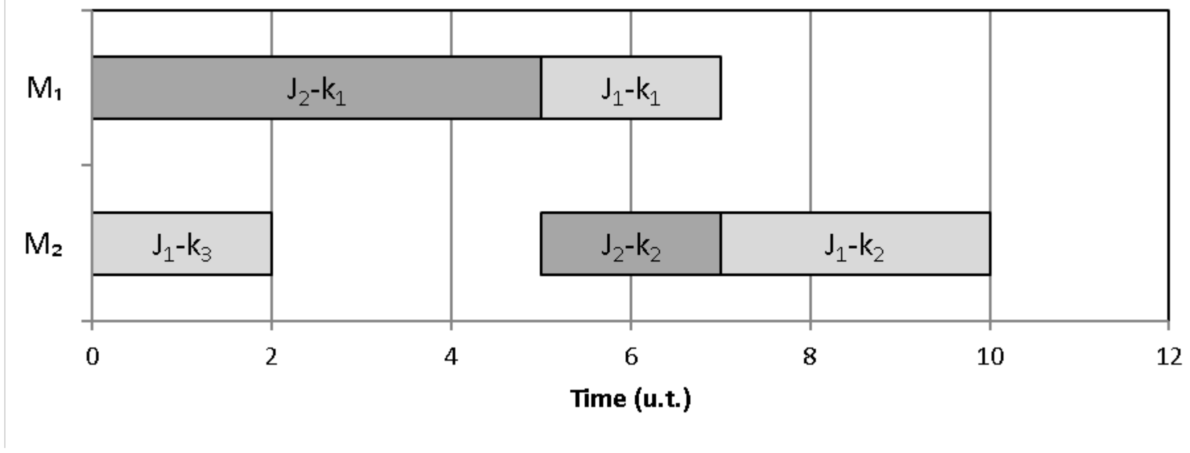
8.3 Problem statement

8.3.1 An illustrative example

The following is an illustrative example of the proposed open shop variant. Let M a set of m machines, N a set of n jobs and O_j a set of operations for each job j with index $k_1, k_2, ..., k_{|O_j|}$, one can present a matrix with the processing times for each operation, per job, in each machine. In the notation of Lawler *et al.* (1993), the problem is defined as: $O_m/rcrc/C_{max}$. Table 30 provides an illustrative example of instance with $m = 2$, $n = 2$, three operations for J_1 ($O_1 = \{k_1, k_2, k_3\}$) and two operations for J_2 ($O_2 = \{k_1, k_2\}$), where p_{ij}^k is the processing time for the operation k of job j in machine i. Figure 60 illustrates an Gantt Chart of a feasible solution for the problem under study. In this example, each job presents a different amount of required operations with a makespan of 10 units of time (u.t.).

One can observe in Figure 60 that the first job was processed two times in the second machine, for different operations (in this case, operations 3 and 2, respectively). Furthermore, all jobs have to be processed in each machine at least one time. The operations of jobs in machines can be performed in any sequence.

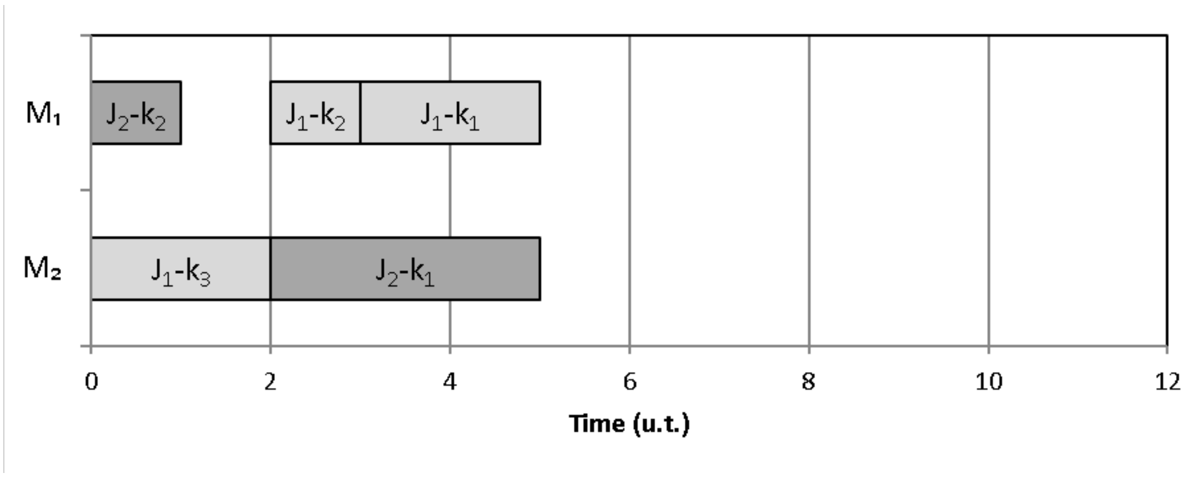
Figure 60 – Gantt chart for the solution (makespan = 10 u.t.)



Source: Authors

Figure 61 illustrates the optimal solution for the presented instance, with the lowest makespan of 5 u.t. The finishing times of operation 1 of jobs 1 and operation 1 of job 2 end in sync, contributing to reduce the makespan. There is just idle time of 1 u.t. in machine 1 for these jobs operations scheduling, whereas in Figure 60 there is a high idle time of 3 u.t. in machine 2 between operation 3 of job 1 and operation 2 of job 2, contributing to increase the makespan.

Figure 61 – Gantt chart for the best solution (makespan = 5 u.t.)



Source: Authors

8.3.2 Mixed-integer linear programming model

Although mixed-integer programming models are usually not efficient methods for solving large-sized instances of many production scheduling problems due to their NP-hard characteristic, it find useful to present a mathematical programming model for the problem to assess the quality of the heuristics proposed for small-sized instances where the optimal can be found. Hereafter, the notation used for the problem is presented.

Indices and sets:

$i \in M$: index for machines.

$j, l, e \in N$: indices for jobs.

$k, w, q \in O_j$: indices for operations of job j.

Parameters:

p_{ij}^{k}: processing time of job j in machine i for the operation k.

$\mathcal{M}$: a large and positive number.

Decision variables:

C_{max}: makespan.

x_{ij}^{k}: 1, if operation k of job j is processed on machine i, and 0 otherwise.

s_j^k: starting time of operation k of job j.

y_{ile}^{wq}: 1, if operation w of job l precedes operation q of job e in machine i, and 0 otherwise.

z_{ile}^{wq}: 1, if operation w of job l and operation q of job e are processed in machine i, and 0 otherwise.

b_j^{wq}: 1, if operation w of job j precedes operation q of job j, and 0 otherwise.

The mixed-integer linear programming model for the OSSPR is presented bellow:

minimize

$$C_{max} \tag{8.1}$$

subject to

$$\sum_{i \in M} x_{ij}^k = 1 \qquad j \in N; k \in O_j \tag{8.2}$$

$$\sum_{k \in O_j} x_{ij}^k \geq 1 \qquad i \in M; j \in N \tag{8.3}$$

$$\left.\begin{aligned} &s_e^q \geq s_l^w + p_{il}^w - \mathcal{M}(2 - z_{ile}^{wq} - y_{ile}^{wq}) \\ &s_l^w \geq s_e^q + p_{ie}^q - \mathcal{M}(1 - z_{ile}^{wq} + y_{ile}^{wq}) \\ &z_{ile}^{wq} \geq y_{ile}^{wq} \end{aligned}\right\} \qquad \begin{aligned} &l, e \in N; w \in O_l; \\ &q \in O_e; i \in M; \\ &w \neq q \text{ if } l = e \end{aligned} \tag{8.4}$$

$$\left.\begin{aligned} &z_{ile}^{wq} \geq x_{il}^w + x_{ie}^q - 1 \\ &z_{ile}^{wq} \leq x_{il}^w \\ &z_{ile}^{wq} \leq x_{ie}^q \end{aligned}\right\} \qquad \begin{aligned} &l, e \in N; w \in O_l; \\ &q \in O_e; i \in M; \\ &w \neq q \text{ if } l = e \end{aligned} \tag{8.5}$$

$$\left.\begin{aligned} &s_j^w \geq s_j^q + \sum_{i \in M} p_{ij}^q x_{ij}^q + \mathcal{M}(1 - b_j^{wq}) \\ &s_j^q \geq s_j^w + \sum_{i \in M} p_{ij}^w x_{ij}^w + \mathcal{M} b_j^{wq} \end{aligned}\right\} \qquad \begin{aligned} &j \in N; w, q \in O_j; \\ &w \neq q \end{aligned} \tag{8.6}$$

$$C_{max} \geq s_j^k + x_{ij}^k p_{ij}^k \qquad i \in M; j \in N; k \in O_j \tag{8.7}$$

$$x_{ij}^k, y_{ile}^{wq}, z_{ile}^{wq}, b_j^{wq} \in \{0, 1\} \qquad \begin{aligned} &i \in M; j, l, e \in N \\ &k \in O_j; w \in O_l; q \in O_e \end{aligned} \tag{8.8}$$

$$s_j^k \in \mathbb{R}^+ \qquad j \in N; k \in O_j \tag{8.9}$$

$$C_{max} \in \mathbb{R}^+ \tag{8.10}$$

The objective function (1) is the makespan minimization. The set of constraints (8.2) guarantees that each operation is processed by exactly one machine. The set of constraints (8.3) guarantees that each machine processes every job at least once. The set of constraints (8.4) imposes that a given machine cannot process two or more jobs simultaneously. Constraint sets (8.5) imposes that variables z_{ile}^{wq} assume value 1 if and only if both x_{il}^w and x_{ie}^q assume value 1. Constraint sets (8.6) impose that operations of the same job cannot be processed simultaneously. Constraint sets (8.7) calculate the makespan. Finally, constraint sets (8.8), (8.9) and (8.10) define the scope of decision variables.

8.3.3 Constraint programming model

Constraint Programming (CP) is a recent paradigm to solve combinatorial optimization problems, particularly for complex problems that are not solved easily with integer linear Equations (ROSSI; BEEK; WALSH, 2006). CP appeared initially in the artificial intelligence area; however, it has been presented high-quality results when applied to solve production sequencing problems (PINEDO, 2016).

CP Optimizer is a constraint programming solver that extends the concepts of classical CP to production scheduling problems. The CP optimizer has a structure to handle complex production scheduling constraints and has an automatic search algorithm that uses metaheuristics in its execution. The CP model developed in this article uses the modeling notation of the CP Optimizer. For more information, see (LABORIE *et al.*, 2018) and the CP Optimizer manual.

The problem under study can be modeled with CP using sets, parameters, and two types of domains: interval decision variables and interval sequence decision variables. The first one represents an operation from a given job to be processed in a given machine, with start and finish times. The second one represents several interval decision variables, such as the processing sequence. Next, there is the proposed CP model for the OSSPR.

Indices and sets:

$i \in M$: index for machines.

$j \in N$: index for jobs.

$k \in O_j$: index for operations of job j.

Parameters:

p_{ij}^k: processing time of job j in machine i for the operation k.

Decision variables:

x_{ij}^k: optional interval variable for processing the operation k of job j in machine i.

y_{jk}: interval variable for to indicate the operation k of job j.

Γ_i: sequence variable with order of x_{ij}^k interval variable in machine i.

The constraint programming model for the OSSPR is presented bellow:

$$\text{minimize}$$

$$\max_{i \in M : j \in N : k \in O_j} \text{endOf}\left(x_{ij}^k\right) \tag{8.11}$$

subject to

$$\text{noOverlap}\left(\Gamma_i\right) \qquad i \in M \qquad (8.12)$$

$$\text{noOverlap}\left(\left[x_{ij}^k\right]_{i\in M:k\in O_j}\right) \qquad j \in N \qquad (8.13)$$

$$\text{alternative}\left(y_{jk}, \left[x_{ij}^k\right]_{i\in M}\right) \qquad j \in N; k \in O_j \qquad (8.14)$$

$$\sum_{k\in O_j} \text{presenceOf}\left(x_{ij}^k\right) \geq 1 \qquad i \in M; j \in N \qquad (8.15)$$

$$\text{interval } x_{ik}^k, \quad \text{opt}, \quad \text{size} = p_{ij}^k \qquad i \in M; j \in N; k \in O_j \qquad (8.16)$$

$$\text{interval } y_{jk} \qquad j \in N; k \in O_j \qquad (8.17)$$

$$\text{sequence } \Gamma_i, \quad \text{on } \left[x_{ij}^k\right]_{j\in N:k\in O_j} \qquad i \in M \qquad (8.18)$$

Equation (8.11) is the makespan minimization. Constraint set (8.12) states that a single job at a time can be produced on machine i. Constraint set (8.13) imposes that a given job j cannot be processed simultaneously for two or more machines. Constraint set (8.14) enforces that a single machine processes a given operation. Constraint set (8.15) enforces that a machine i processes a job j at least once in any operation of this job. Finally, constraints (8.16)-(8.18) define the scope of decision variables. The variables x_{ij}^k and y_{jk} are an intervals decisions variables with a duration, and Γ_i is a variable that stores the sequence of operations on machine i.

8.3.4 Comparison of models

There are several ways to model OSSPRM using mathematical programming. The main ones differ concerning the definition of decision variables, with the notation in three types: positional, sequential, and time-indexed notation. Naderi *et al.* (2011b) show models with sequential notation perform better on OSSP problems due to their smaller number of variables and constraints compared to positional and time-indexed notation.

The MILP model presented in Equations (8.1) - (8.10) uses the positional notation for the decision variables, but due to characteristics of the problem domain, such as the number of operations, it presents a expressive number of constraints, the constraint with the largest size is in Equations (8.4) and (8.5) with worst case complexity (all jobs with the largest number of operations $|O_j|$) of $\mathcal{O}\left(n^2 m |O_j|^2\right)$. The MILP model necessitates $n^2|O_j|^2\left(6m+1\right)+n\left(|O_j|+m+|O_j|m\right)$ constraints and $mn|O_j|\left(1+n\right)+n|O_j|+1$ decision variables, where $mn|O_j|\left(1+n\right)$ are integers and $n|O_j|+1$ are reals.

The CP model uses logical constraints for the problem and applies heuristic techniques to reduce the search space. In addition, constraint programming finds feasible solutions with low computational cost due to exploiting the combinatorial problem domain (ROSSI; BEEK; WALSH, 2006). The use of CP results in a significant facility for modeling combinatorial problems such as production scheduling and other industrial problems, mainly due to the use of logical constraints in modeling (see examples in Lunardi *et al.* (2020), Sacramento, Solnon and Pisinger (2020), Kizilay and Çil (2020), and Pınarbaşı (2021)).

The CP model presented in Equations (8.11) - (8.18) uses the notation of the IBM CP Optimizer solver for scheduling problems, the constraint with the largest size is present in Equations (8.14) and (8.15) with worst case complexity (for $m = n$ see Taillard (1993) and with all jobs with the largest number of operations $|O_j|$) of $\mathcal{O}\left(n|O_j|\right)$. The CP model necessitates $m + n\left(m + |O_j| + 1\right)$ constraints and $m + n\left(m|O_j| + |O_j| + 1\right)$ decision variables, all decision variables are discrete with type interval or sequence. Therefore, due to the properties and modeling forms of CP, it has less constraints and decision variables than MILP, which can contribute to better performance solving the OSSPRM. The Table 31 illustrates a comparison between the key characteristics of exact approaches, for several different instance sizes. With m the number of machines and n the number of jobs.

Table 31 – Comparison of MILP and CP formulations with examples of instances sizes for OSSPR

Instances sets		MILP			CP	
m	n	# integer variables	# continuous variables	# constraints	# integer variables	# constraints
3	3	108	10	1584	**42**	**24**
4	4	320	17	6496	**88**	**40**
5	5	750	26	19550	**160**	**60**
6	6	1512	37	48240	**264**	**84**
7	7	2744	50	103684	**406**	**112**
8	8	4608	65	201344	**592**	**144**
9	9	7290	82	361746	**828**	**180**
10	10	11000	101	611200	**1120**	**220**
15	15	54000	226	4610700	**3630**	**480**
20	20	168000	401	19368800	**8440**	**840**

Source: Authors.

Analyzing the Table 31, it can be observed that with the increase in the size of the problem, the decision variables and restrictions of the MILP models have a significant increase when compared to the variables and restrictions of the CP models.

8.3.5 Problem properties

In this subsection, are presented two properties for the OSSPR. Firstly, is presented the NP-hardness of the proposed problem. Secondly, is presented a lower bound for the OSSPR.

Theorem 8.3.1. *The OSSPR is NP-hard.*

Proof. Taking into consideration the set of operations of jobs O_j, if $O_j = \{k_1, k_2, \ldots, k_m\}$, $\forall j \in N$ and $p_{ij}^k = p_{ij}$, $\forall i \in M; j \in N; k \in O_j$, each job have just one operation in each machine and the problem is equivalent of the classic OSSP, which is NP-Hard for $m \geq 3$ (GONZALEZ; SAHNI, 1976). Thus, the classic OSSP is a particular case of the OSSPR. □

The classical open shop is an NP-hard problem (GONZALEZ; SAHNI, 1976), thus the variant under study also is NP-hard, because if there are not repetitions, the problem can be reduced to the classical open shop. Therefore, there is no guarantee that mixed-integer linear programming models, as well as heuristic algorithms, can find the global optimal solution, or even high-quality solutions. In this sense, the determination of a lower bound, aiming at obtaining a reference for the evaluation of a given solution found, is quite relevant.

Proposition 8.3.2. *A lower bound for OSSPR is given by:*

$$LB = \max\left\{\max_{j\in N}\left\{\sum_{i\in M}\left(\min_{k\in O_j}\left\{p_{ij}^k\right\}\right) + (|O_j| - m) \times \min_{i\in M: k\in O_j}\left\{p_{ij}^k\right\}\right\}, \max_{i\in M}\left\{\sum_{j\in N}\min_{k\in O_j}\left\{p_{ij}^k\right\}\right\}\right\} \tag{8.19}$$

Proof. The lower bound considers there is no idle time between operations of jobs in machines. Taking into basis the well-known lower bound for the classic open shop present by Pinedo (2016), this study extend this concept to the OSSPR. The lower bound considers the greatest value between the summation of processing times per job plus the summation of processing times of the remaining operations per job ($|O_j| - m$) and the summation of processing times per machine. If $|O_j| = m$ $\forall j \in N$ the lower bound reduces to a lower bound of classic OSSP. □

8.4 Variable search strategy with constraint programming

In the last few years, several contributions have presented hybridization of mathematical programming and heuristics to solve production sequencing problems (CROCE; NARAYAN; TADEI, 1996; LIN; YING, 2016; FRAMINAN; PEREZ-GONZALEZ, 2018; PRATA; RODRIGUES; FRAMINAN, 2021; PRATA; ABREU; LIMA, 2020). Another promising research topic is the application of constraint programming approaches for the resolution of production sequencing problems. We can observe that the hybridization of constraint programming and metaheuristics is rather limited, despite the potential benefits of this class of algorithms (ALAMEEN *et al.*, 2016; HOJABRI *et al.*, 2018; HÀ *et al.*, 2020).

Firstly proposed by Mladenović and Hansen (1997), the variable neighborhood search (VNS) is a metaheuristic based on systematic changes in the neighborhood structure, aiming to escape from local optima to solve optimization problems. VNS explores different neighborhoods of the incumbent solutions and moves to a new solution taking into account the improvement of the current incumbent solution. The VNS has received a lot of attention from researchers in the production scheduling area mainly due to its low dependency on parameters and robustness (HANSEN; MLADENOVIĆ; PÉREZ, 2010).

Based on the above, we presented the proposition of a hybrid algorithm considering constraint programming and variable neighborhood search methods. Instead of the neighborhood concept, it adopts the conception of search strategy in constraint programming. Starting from an initial heuristic solution, several search strategies with a given time limit are applied, accepting solutions with a better makespan value. The proposal explores different search strategies in a constraint programming model, escaping from local optima and reducing the computation effort using different search strategies.

Concerning the terminology of a VNS framework, the proposed VNS-CP algorithm is a Union VNS approach (HANSEN *et al.*, 2017). The proposed Union VNS explores, in each iteration, one type of neighborhood, chosen randomly from the set of all neighborhoods of the solution. In addition, Union VNS-CP considers the set of CP model search strategies as the union of the neighborhoods in the iterations of the algorithm.

Among the main constraint programming solvers, the CP Optimizer, which presents an extension of the classic CP models, considering logical expressions to modeling complex scheduling constraints (LABORIE, 2018). CP Optimizer presents several different search strategies to define before the solver execution; for example: Auto: presents a hybridization between exact and Self-Adapting Large-Neighborhood Search algorithms. Depth-first: a well-known tree search algorithm. Restart: the constructive search is restarted periodically and guided to the optimal solution. MultiPoint: generate a set of solutions that are combined, aiming to produce better solutions. The Auto and Restart strategies can provide the optimally proof. On the other hand, the Depth-first and Multipoint strategies cannot provide this proof due to the heuristic nature of the solver. The hybrid VNS consider the four strategies as possible local searches.

Each search strategy can generate good results, depending on the evaluated test instances and available computational time. According to Apt (2003), constraint programming uses logic programming and constraint solving techniques to create solutions to optimization problems, such as Boolean satisfiability problems (SAT).

The hybrid algorithm uses two representations throughout the search process. The first one is the constraint programming encoding, expressed by decision variables x, y, and Γ. The second one is heuristic encoding, using a permutational representation of the operations Naderi *et al.* (2010). Each operation is allocated in the earliest available machine, considering the constraint in which the jobs are processed at least once in each machine.

For sequence representation, a $\sum_{j\in N}^{n} |O_j|$ array to represent a sequence that generates a solution is used, each bit from the sequence contains values $= 1, \dots, \sum_{j\in N}^{n} |O_j|$, where the $\hat{k}$-th operation of the $\hat{j}$-th job is represented by the value $\sum_{j=1}^{\hat{j}-1} |O_j| + \hat{k}$. Figure 62 illustrates an example of a possible sequence for 2 jobs and 3 operations of job 1 and 2 operations of job 2.

Figure 62 – An illustrative example for the sequence encoding.

Sequence representation:	1	2	3	4	5
Operation representation:	J_1-k_1	J_1-k_2	J_1-k_3	J_2-k_1	J_2-k_2
Sequence solution:	3	5	2	4	1
Decoded solution:	J_1-k_3	J_2-k_2	J_1-k_2	J_2-k_1	J_1-k_1

Source: Authors

Algorithm 63 describes the proposed Variable Neighborhood Search Constraint Programming (VNS-CP) solution procedure. The inputs of the algorithm are the processing times matrix, the number of iterations *niter*, the time limit TL, and the set of search strategies $\mathcal{Q}$. The function $CREATE\_CP\_SOLUTION(SOL, p)$ converts the heuristic encoding representation of the sequence of operations to the constraint programming encoding representation for use in the CP model. The inverse function is $CREATE\_SEQUENCE\_SOLUTION(x, y, \Gamma, p)$ which converts the constraint programming encoding representation to the heuristic encoding representation. Finally, the function $RUN\_CP(x, y, \Gamma, search_{strategy} = st, time_{limit} = TL)$ performs a CP Optimizer execution with the search strategy st and the time limit TL seconds, returning a solution SOL already in the heuristic encoding representation.

The initial solution is the MIH constructive heuristic, which was proposed by Abreu *et al.* (2020) for the open shop with sequence-dependent setup times with the objective function of minimizing the total completion time. Next, this heuristic solution is converted into a CP solution. While the algorithm's runtime is not over, a CP search strategy is randomly selected and executed within the time limit TL. If the solution is improved, it is used as the new incumbent solution. In each iteration, the GAP as the relative deviation between the best solution found and the lower bound is calculated. If this value is less than or equal to 10^{-4} (the considered tolerance), the search is finished with the global optimal solution found. Otherwise, the algorithm is performed until the available run time ends.

```
   Data: Instance $p$, $niter$, $TL$, $\mathcal{Q}$
   Result: Solution $\mathcal{S}$
 1 $\mathcal{S} \leftarrow MIH(p)$;
 2 $makespan\_best \leftarrow makespan(\mathcal{S})$;
 3 $x, y, \Gamma \leftarrow CREATE\_CP\_SOLUTION(SOL, p)$;
 4 $iter \leftarrow 0$;
 5 while run time does not end do
 6     $st \leftarrow$ chosen a random search strategy $\in \mathcal{Q}$;
 7     $x', y', \Gamma'$, GAP $\leftarrow RUN\_CP(x, y, \Gamma, search_{strategy} = st, time_{limit} = TL)$;
 8     $\mathcal{S}' \leftarrow CREATE\_SEQUENCE\_SOLUTION(x', y', \Gamma', p)$;
 9     $makespan\_new \leftarrow makespan(\mathcal{S}', p)$;
10     $\Delta \leftarrow makespan\_new - makespan\_best$;
11     if $\Delta < 0$ then
12         $makespan\_best \leftarrow makespan\_new$;
13         $x, y, \Gamma \leftarrow x', y', \Gamma'$;
14         $\mathcal{S} \leftarrow \mathcal{S}'$;
15         if the solution GAP is less than or equal to $10^{-4}$ then
16             break while loop;
17         end
18     end
19 end
```

Figure 63 – VNS-CP

8.5 Computational experience

8.5.1 Proposed test instances and performance criteria

Since the problem under study was not previously reported in the revised literature, three sets of test instances are proposed, based on the well-known benchmark problems for the classic open shop proposed by Guéret and Prins (1998), Brucker *et al.* (1997), and Taillard (1993).

In the test problems proposed by Guéret and Prins a fixed interval for the processing times is considered, with random values uniformly distributed between 1 and 1000, and a constant value for the lower bound equal to 1000. Different problem sizes are considered with $n, m \in \{3, 4, 5, 6, 7, 8, 9, 10\}$. For each class were randomly generated 10 test instances, totaling 80 instances. Brucker et al. test problems are generated in a similar way, with random values uniformly distributed between 1 and 500 and 6 sets of jobs $n, m \in \{3, 4, 5, 6, 7, 8\}$, totaling 60 instances. Taillard test problems were generated with random values uniformly distributed between 1 and 100, without a lower bound constraint. Problem classes were considered according to the combination of the 6 sets of jobs $n, m \in \{4, 5, 7, 10, 15, 20\}$. For each size, 10 instances are randomly generated, totaling 60 instances. For the OSSPR every instance of Guéret and Prins, Brucker et al., and Taillard were adapted adding randomly a number of repetitions between 0 and 2 to each job regarding to at least one job to be repeated at least once.

Relative percentage deviation (RPD) is the performance criteria used in this article. Equation (8.20) shows the calculation of RPD. In this equation, sol_{method} represents the solution returned by a given method, and LB the lower bound calculated by Equation (8.19).

$$RPD = \frac{sol_{method} - LB}{LB} \times 100 \tag{8.20}$$

For the mixed-integer linear programming model implementation, the solver IBM ILOG CPLEX 12.10 with C++ programming language is used. The LTPROM, MIH, IRH1 to IRH4, and EGA methods were implemented in the Intel®Distribution for Python (https://software.intel.com/en-us/distribution-for-python) and were run in C with Cython library (http://cython.org/) (BEHNEL *et al.*, 2011). The constraint programming model and VNS-CP were developed with the python programming language using DOcplex 2.10.155 libray with IBM ILOG CP Optimizer 12.10. The computational experience was performed on a Linux Ubuntu 18.04 64bits machine with 8GB of memory and Intel Core i5-3470 CPU 3.20 GHz ×4 processor.

The source codes, instance sets, results of all computational tests, and statistical analyses are available in the following link: http://repositorio.uspdigital.usp.br/handle/item/444. Another data or any questions are available upon request.

8.5.2 Test results for exact methods

For the exact models MILP and CP models, a time limit of 600s is adopted in computational tests. The considered methods are listed bellow. Four search strategies of CP Optimizer solver are tested.

- Mixed Integer Linear Programming (MILP): the proposed mathematical programming model given by the Equations (8.1)-(8.9).
- Constraint Programming Auto (CP Auto): the proposed CP model given by the Equations (8.11)-(8.18) with automatic search strategy.
- Constraint Programming Depth-first (CP DF): the proposed CP model with Depth-first search strategy.
- Constraint Programming Restart (CP RS): the proposed CP model with Restart search strategy.
- Constraint Programming Multi-Point (CP MP): the proposed CP model with Multi-Point search strategy.

Table 32 illustrates the Average Relative Percentage Deviation (ARPD) and Average Computational Times (Avg. CT (s)) for each exact method evaluated in each set of instances. The last four lines provide a summary of the computational results: the best ARPD value (Min), the average ARPD (Average), the worst ARPD value (Max), as well as the number of unsolved instances. The results marked "***" indicate that the tested method did not find any feasible solution for all instances of the set within the time limit.

Table 32 – ARPD results and average computational times of exact methods in all sets of instances

Group	Size	MILP		CP Auto		CP DF		CP RS		CP MP	
		ARPD	Avg. CT (s)	ARPD	Avg. CT (s)	ARPD	Avg. CT (s)	ARPD	Avg. CT (s)	ARPD	Avg. CT (s)
Gueret and Prins	3	**1.47**	1.17	**1.47**	10.85	**1.47**	600.03	**1.47**	10.60	**1.47**	600.02
	4	**24.54**	548.44	**24.54**	473.88	24.98	600.03	**24.54**	396.40	**24.54**	600.03
	5	26.04	600.00	**25.07**	596.93	67.12	600.04	**25.07**	600.07	**25.07**	600.04
	6	45.14	600.00	**26.45**	600.13	***	600.00	**26.45**	600.11	**26.45**	600.05
	7	100.35	600.00	**19.24**	600.17	***	600.00	19.26	600.16	**19.24**	600.06
	8	***	600.00	14.21	600.20	***	600.00	**12.87**	600.15	14.12	600.07
	9	***	600.00	**13.86**	600.18	***	600.00	***	600.00	16.15	600.10
	10	***	600.00	***	600.00	***	600.00	***	600.00	***	600.00
Taillard	4	**0.25**	435.60	5.14	19.38	7.95	600.03	5.14	13.59	5.14	600.03
	5	3.47	475.79	**0.00**	10.24	***	600.00	**0.00**	7.36	**0.00**	600.03
	7	133.08	600.00	**0.00**	547.07	***	600.00	**0.00**	495.40	**0.00**	600.06
	10	***	600.00	**1.81**	600.08	***	600.00	***	600.00	1.93	600.09
	15	***	600.00	***	600.00	***	600.00	***	600.00	***	600.00
	20	***	600.00	***	600.00	***	600.00	***	600.00	***	600.00
Brucker	3	**5.07**	2.82	**5.07**	2.48	**5.07**	600.03	**5.07**	2.39	**5.07**	600.03
	4	**3.69**	600.00	**3.69**	281.85	5.18	600.03	**3.69**	279.17	**3.69**	600.03
	5	11.22	600.00	**1.57**	489.24	8.48	600.04	**1.57**	482.94	1.61	600.04
	6	59.37	600.00	**1.10**	478.03	***	600.00	**1.10**	536.54	1.48	600.04
	7	121.40	600.00	1.10	600.07	***	600.00	**1.07**	600.08	2.42	600.06
	8	***	600.00	2.76	600.11	***	600.00	**2.75**	600.11	3.66	600.07
Min		0.25	1.17	**0.00**	2.48	1.47	600.00	**0.00**	2.39	**0.00**	600.00
Average		41.16	523.19	**8.65**	445.54	17.18	600.01	8.67	441.25	8.94	600.04
Max		133.08	600.00	**26.45**	600.20	67.12	600.04	**26.45**	600.16	**26.45**	600.10
Unsolved Instances		73	–	**38**	–	144	–	66	–	42	–

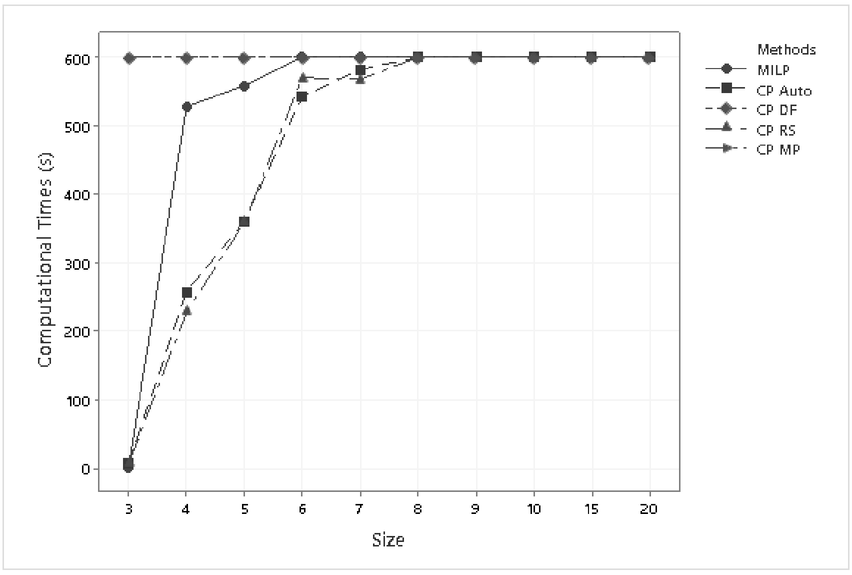
Source: Authors.

Table 32 shows that all the exact methods have not returned feasible solutions for some test instances within the considered time limit. The CP DF and MILP methods presented a greater number of unsolved instances. All the other CP-based methods presented better results, expressed by a lower average ARPD and fewer unsolved instances than MILP. Also, the CP Auto method presented the best values of average ARPD and the number of unsolved instances than all other methods; however, it presented worse ARPD results than CP RS in Brucker instances with sizes 7 and 8. CP MP presented high-quality solutions in the instances of Taillard Gueret and Prins. Furthermore, the CP MP method solved more instances than CP DF and CP RS strategies.

Since no exact methods were able to solve all the evaluated test instances, and the distinct search strategies returned different results among the considered sets of instances, the proposed hybrid method, described in the Section 8.4, is evaluated. The VNS-CP approach adopt the three better search strategies (CP Auto, CP RS, and CP MP).

Figure 64 illustrates the average computational times, clustered by the size of the instances (expressed by the number of jobs). For all the evaluated exact methods, the time limit was reached for the large-sized instances. Among the evaluated exact methods, the CP Auto and the CP RS presented the lowest computational times, only reaching the time limit for the problems with 8 or more jobs. On the other hand, the methods CP MP and CP DF, when did not provide the optimality proof, were executed using the specified time limit.

Figure 64 – Average computational times for all proposed exact methods in each instance size



Source: Authors

The CP RS and CP DF have a higher computational time than MILP, but overall, they get a better average ARPD than MILP. Also, the CP RS, CP Auto, and CP MP methods can find feasible solutions for more instances than MILP.

8.5.3 Test results for approximate methods

Concerning the approximate methods under comparison, we can highlight the following issues. Since the problem under study is not reported yet in the current literature, the proposed approaches (the MILP model, CP model and the VNS-CP algorithm) cannot be directly compared with other methods. This article considers the adaptation of three constructive heuristics: a classic priority rule, a recent heuristic and a competitive meta-heuristic for classic OSSP.

These three approaches are inserted as a complementary reference (the lower bound is a basis for the comparisons). The MIH is used in initial solution for the VNS-CP as a hybrid solution approach, it is possible to evaluate the improvement provided by the VNS with CP procedure.

The considered methods are listed bellow. Eight approximate methods are tested to solve the OSSPR.

- Longest Total Remaining Processing Times on Other Machines first (LTRPOM): a adaptation of constructive heuristic developed by Pinedo (2016). It is a more general rule than classical priority rule LAPT also developed by Pinedo (2016) for the OSSP.
- Minimal Idleness Heuristic (MIH): a constructive heuristic developed by Abreu *et al.* (2020) and adapted for OSSPR.
- Insertion and Reinsertion Heuristic 1-4 (IRH1 to IRH4): local search algorithms proposed by Naderi *et al.* (2010) and adapted for OSSPR.
- The extended genetic algorithm (EGA): is a adaptation of the most recent metaheuristic for the classic OSSP proposed by Hosseinabadi *et al.* (2018).
- A variable neighborhood search with constraint programming with a variable search strategy (VNS-CP): The new approach to solve the OSSPR.

The computational times of two constructive heuristics LTPROM and MIH are negligible (less than one second for all the evaluated test instances). The IRx heuristics are constructive algorithms that present variable execution times, which are proportionate to the size of the instances. The EGA and the VNS-CP were run with a time limit of 600s. After several preliminary computational experiments, the time limit for the CP Optimizer execution TL is set in 200s. For the EGA, the same parameters used by Hosseinabadi *et al.* (2018) were considered in the tests: population size 92, crossover ratio 0.8, and mutation ratio 0.2.

Table 33 illustrates the ARPD values for each method under comparison for each set of instances. As we have already defined previously, the last four lines are related to the summary of the results.

Due to the stochastic behavior of VNS-CP and EGA, each method was run five times for each instance. Table 33 shows the best and average results of the ARPD of both algorithms in each set of instances. For comparison purposes between exact and approximation methods, the Table 33 shows the best result found by all exact approaches in each set of instances. The results marked *** indicate that the tested method did not find any feasible solution for all instances of the set within the time limit. Analyzing Table 33, it can observe that feasible solutions are found for all the methods under comparison for all the test instances.

VNS-CP uses the best search strategies tested. The metaheuristic can find feasible solutions for all instances tested, which shows a good improvement with the hybridization

Table 33 – ARPD results of approximate methods in all sets of instances

Group	Size	Best Exact Results	LTRPOS	MIH	IR1	IR2	IR3	IR4	EGA		VNS-CP	
									Best	Avg.	Best	Avg.
Gueret and Prins	3	**1.47**	39.39	43.41	8.00	8.00	4.35	7.75	1.58	3.29	**1.47**	**1.47**
	4	**24.54**	89.65	88.38	49.84	49.84	28.73	30.23	26.28	29.02	**24.54**	**24.54**
	5	**25.07**	115.66	82.16	44.88	44.88	29.18	39.28	26.53	28.78	**25.07**	**25.07**
	6	**26.45**	124.62	82.86	48.13	48.13	33.80	35.95	28.11	30.75	**26.45**	**26.45**
	7	**19.24**	129.36	83.84	55.10	55.10	35.37	42.30	21.29	24.06	**19.24**	**19.24**
	8	**12.87**	152.05	72.60	59.11	59.11	39.85	40.08	20.28	22.89	13.87	13.90
	9	13.86	185.03	68.32	61.49	61.49	35.53	44.82	18.63	20.63	**11.61**	11.78
	10	***	181.90	55.39	56.01	56.01	36.48	42.77	20.62	23.38	**8.49**	8.99
Taillard	4	0.25	54.64	26.43	13.49	13.49	5.49	6.70	1.53	4.08	**0.11**	**0.11**
	5	**0.00**	64.46	24.37	11.10	11.10	6.07	5.96	0.96	3.57	**0.00**	**0.00**
	7	**0.00**	79.61	15.20	19.17	19.17	7.70	7.58	0.68	2.67	**0.00**	**0.00**
	10	1.81	101.16	14.14	26.65	26.65	15.99	13.26	4.19	6.39	**0.00**	0.31
	15	***	109.77	10.14	37.94	37.94	30.54	17.85	9.55	12.07	**0.00**	**0.00**
	20	***	124.51	9.18	46.15	46.15	40.96	25.91	13.03	15.80	**0.00**	0.40
Brucker	3	**5.07**	37.59	48.08	19.29	19.29	12.17	11.92	8.58	10.85	**5.07**	**5.07**
	4	**3.69**	59.01	37.59	27.15	27.15	16.87	18.08	8.86	11.62	**3.69**	**3.69**
	5	**1.57**	84.75	40.52	29.95	29.95	13.61	15.45	5.83	8.15	**1.57**	**1.57**
	6	**1.10**	116.85	44.92	28.93	28.93	16.70	23.35	5.90	8.34	**1.10**	1.24
	7	**1.07**	143.64	35.05	43.55	43.55	26.26	27.48	8.81	10.97	1.09	1.26
	8	2.75	161.06	42.17	41.97	41.97	22.10	29.87	9.66	11.82	**1.35**	1.47
Min		**0.00**	37.59	9.18	8.00	8.00	4.35	5.96	0.68	2.67	**0.00**	**0.00**
Average		8.28	107.74	46.24	36.40	36.40	22.89	24.33	12.04	14.46	**7.24**	7.33
Max		**26.45**	185.03	88.38	61.49	61.49	40.96	44.82	28.11	30.75	**26.45**	**26.45**
Unsolved Instances		37	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**

Source: Authors.

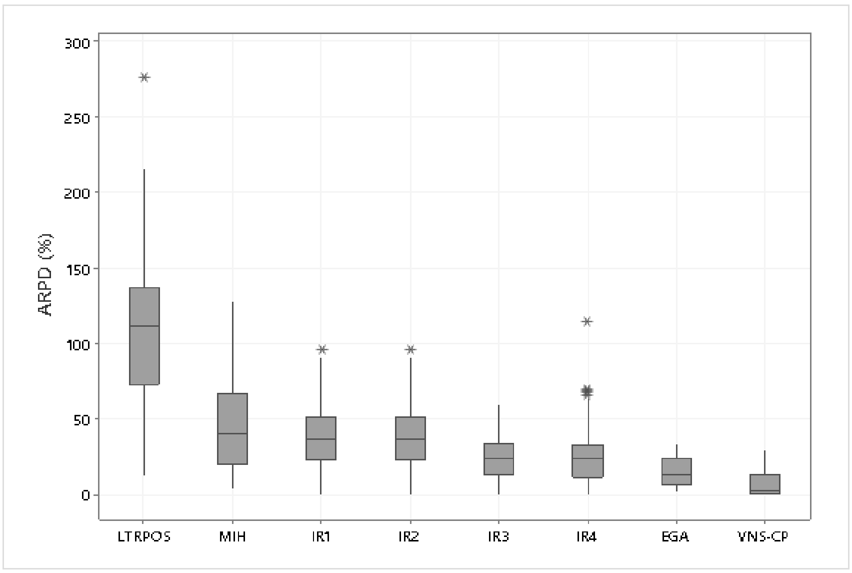
of exact and approximate techniques. Furthermore, the VNS-CP returned the best values for Min, Average, and Max performance indicators. Comparing the best results with the average results of the VNS-CP, both have similar values, which might indicate that the VNS-CP has low variability of results.

Figure 65 illustrates the box plot with ARPD distribution for all proposed approximate methods. LTRPOS and MIH get the worst median results, IRx heuristics get the average performance of median of ARPD. EGA and VNS-CP get the smallest median of ARPD, with the VNS-CP obtaining the best median and interquartile range between all approximate methods.

Table 34 shows the average computational times of all approximate approaches in each set of instances. The constructive heuristics LTRPOS and MIH have ineligible computational times. The IR-based heuristics are constructive algorithms with local search procedures that present an exponential increase in computational time with an increase in instance size, especially IR3 and IR4. It can be observed that the IR-based heuristics are deterministic.

Based on the results obtained, it can be observed that the EGA and VNS-CP

Figure 65 – Distribution of ARPD for all proposed approximate methods

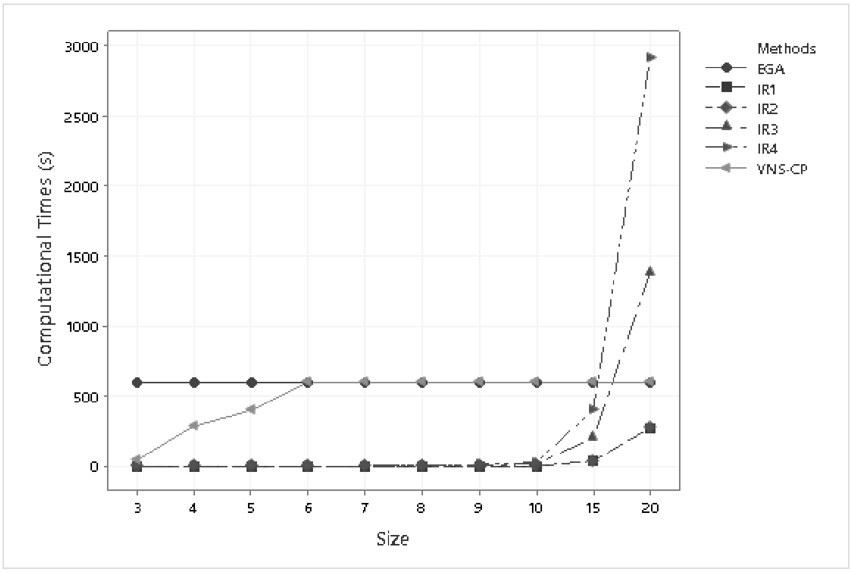


Source: Authors

algorithms presented similar computational times. However, for small and medium-sized instances, VNS-CP can prove optimally, with the algorithm finishing before the time limit.

Figure 66 illustrates the average computational times for each instance size and proposed approximate methods. The time limit could be reached for all the evaluated approximate methods. Besides, the constructive heuristics IRx with local search could exceed this value for the large-sized instances (with 15 and 20 jobs).

Figure 66 – Average computational times for all proposed approximate methods in each instance size



Source: Authors

The VNS-CP (the article proposal) outperformed all the other approximate ap-

Table 34 – Average computational times of approximate methods in all sets of instances

Group	Size	LTRPOS	MIH	IR1	IR2	IR3	IR4	ILS	EGA	VNS-CP
Gueret and Prins	3	<0.01	<0.10	0.01	0.00	0.02	0.02	4.03	600.41	15.75
	4	<0.01	<0.10	0.01	0.01	0.07	0.11	26.30	600.57	547.34
	5	<0.01	<0.10	0.04	0.04	0.20	0.35	27.06	600.42	600.51
	6	<0.01	<0.10	0.13	0.13	0.66	1.12	29.73	600.59	600.45
	7	<0.01	<0.10	0.32	0.32	1.83	2.84	31.67	600.56	600.36
	8	<0.01	<0.10	0.69	0.68	3.60	6.41	36.02	600.50	600.48
	9	<0.01	<0.10	1.42	1.42	8.37	13.27	30.82	600.36	600.47
	10	<0.01	<0.10	2.82	2.81	16.24	26.95	28.39	600.48	600.56
Taillard	4	<0.01	<0.10	0.01	0.01	0.06	0.11	2.23	600.51	25.59
	5	<0.01	<0.10	0.05	0.04	0.20	0.39	3.32	600.56	16.66
	7	<0.01	<0.10	0.29	0.29	1.59	2.61	3.96	600.47	600.46
	10	<0.01	<0.10	2.75	2.75	16.34	27.61	6.23	600.50	600.47
	15	<0.01	<0.10	38.51	38.54	201.04	405.04	3.38	600.60	600.60
	20	<0.01	<0.10	273.13	273.27	1383.88	2926.48	3.73	600.59	600.60
Brucker	3	<0.01	<0.10	0.01	0.00	0.01	0.02	28.39	600.61	78.80
	4	<0.01	<0.10	0.01	0.01	0.06	0.09	6.39	600.44	286.98
	5	<0.01	<0.10	0.04	0.04	0.24	0.38	11.14	600.51	600.44
	6	<0.01	<0.10	0.14	0.13	0.82	1.26	18.45	600.51	600.54
	7	<0.01	<0.10	0.31	0.31	1.92	2.97	20.82	600.48	600.40
	8	<0.01	<0.10	0.68	0.67	4.76	6.69	18.26	600.35	600.53
Min		<0.01	<0.10	0.01	0.00	0.01	0.02	2.23	600.35	15.75
Average		<0.01	<0.10	16.07	16.08	82.09	171.24	17.02	600.50	468.90
Max		<0.01	<0.10	273.13	273.27	1383.88	2926.48	36.02	600.61	600.60

Source: Authors.

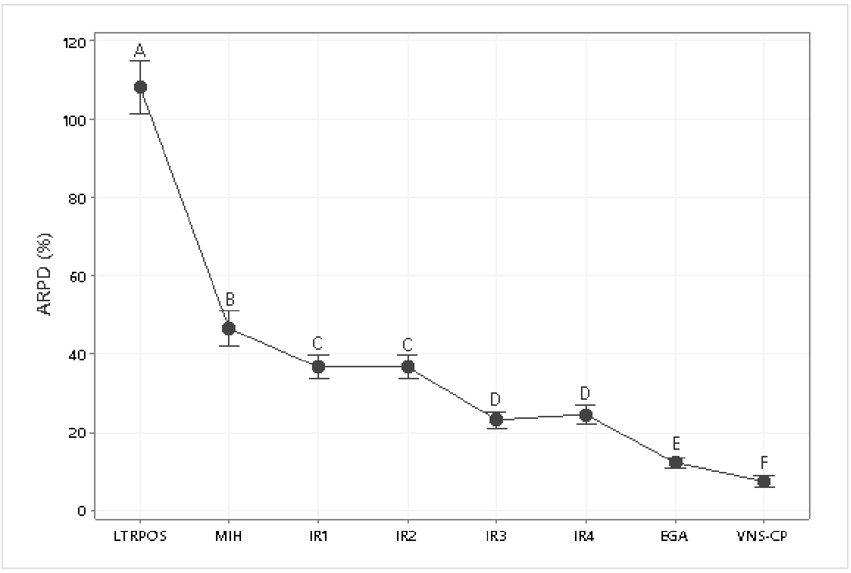
proaches under comparison for ARPD with competitive computation times. Due to the fact that the proposed approach presents optimally proof, the algorithm was able to finish the search before the specified time limit, for the small-sized and medium-sized test instances. For the large-sized instances with sizes 10, 15, and 20, VNS-CP obtained the best results for these instance sets among all other exact and approximate methods.

In order to validate the results, it is important to verify whether the previous differences in the ARPD values are statistically significant. An non-parametric analysis of variance (Welch Test) is applied (MONTGOMERY, 2017). The p-value is very close to zero. Figure 67 show the ARPD boxplot for all constructive heuristics tested with Games-Howell confidence interval ($\alpha = 0.05$) and groups of the similar mean result. Methods that do not share letters are significantly different.

It can see that there are statistically significant differences between the ARPD values among the approximate methods proposed. The VNS-CP gets the best results. With respect to the ARPD, the difference among VNS-CP and competitive EGA is significant because they are clustered in different groups (with F and E letters, respectively).

Its clearly that the hybridization of the VNS for variable search strategies of

Figure 67 – Mean, confidence intervals and Games-Howell post-hoc test at the 95% confidence level for the approximate methods in all sets of instances.



Source: Authors

constraint programming procedure outperforms the all other tested methods, being the differences statistically significant as they are in different groups in Figure 67. Hence, the VNS-CP turns out to be the best approximate approach for the OSSPR with a good trade-off between solution quality and computational times.

8.6 Conclusions

This article address a new open shop scheduling problem with the processing of more than one operation per job in the same machine, the open shop scheduling problem with repetitions (OSSPR). A mixed-integer linear formulation for the OSSPR is presented. Besides, two constructive heuristics is adapted for the problem under study, and a variable neighborhood search with constraint programming search strategy (VNS-CP) meta-heuristic is proposed for solving the problem.

An extensive computational experimentation was performed to assess the performance of the proposed methods. The lower bound deviation percentage was the indicator of the quality of the solutions, and the average computation time was the indicator of the computational effort. Regarding the quality of the solutions, the VNS-CP outperforms the MILP model, requiring less time than this method. The VNS-CP also outperforms all others approximate methods, becomes the best approach to solve OSSPR.

VNS-CP presents an intensive exploitation mechanism with constraint programming local search procedure, with a variable search strategy to improve search diversification, which is one of the main contributions of the research. With these innovative mechanisms, the VNS-CP approach presented smaller ARPD values, and it outperformed all other evaluated algorithms.

As extensions of this work, variable fixing approaches can improve the solutions generated by the presented mathematical model. These solutions can interact with heuristic methods, as in a matheuristic framework. Future studies could also propose other competitive metaheuristics to solve the problem. Another possibility is to consider different objective functions and constraints for the proposed variant, such as total completion time, total tardiness, or travel time constraints.

9 A NEW EFFICIENT BIASED RANDOM KEY GENETIC ALGORITHM FOR OPEN SHOP SCHEDULING WITH ROUTING BY CAPACITATED SINGLE VEHICLE AND MAKESPAN MINIMIZATION

9.1 Introduction

Production scheduling problems have been widely studied as optimization problems due to their varied industrial applications. The open-shop scheduling problem (OSSP) consists of scheduling a set of jobs in a set of machines, where each operation is associated with a processing time and where there is no preset sequence of operation execution. The problem has a practical and theoretical relevance and has received less attention than classical production scheduling problems (ANAND; PANNEERSELVAM, 2016).

As the problem does not have a predefined sequence of operations, the number of viable solutions is significantly higher than the classical production scheduling problems, such as flow shop and job shop. OSSP is suitable for various industrial applications, such as: plastic injections, chemical processes, the oil industry, food production and pharmaceutical production. In the service sector, this problem can be modelled for scheduling medical services, vehicle maintenance, museum visits and telecommunications (GONZALEZ; SAHNI, 1976; LIN; LEE; PAN, 2008; VINCENT; LIN; CHOU, 2010; NADERI; NAJAFI; YAZDANI, 2012; ABREU *et al.*, 2020).

The vehicle routing problem (VRP) concerns the delivery of products to a set of customers through a set of vehicles on an appropriate road network. The problem has different operational constraints to consider real characteristics of distribution systems (TOTH; VIGO, 2002). Several recent articles address optimization problems involving VRP (ZHANG *et al.*, 2020; ALI; CÔTÉ; COELHO, 2020; CHEN *et al.*, 2020; GHANNADPOUR; ZANDIYEH, 2020; MOLINA *et al.*, 2020). OSSP is directly related to VRP because, as the OSSP is a production scheduling problem, it is necessary to schedule the delivery of products to the customers after the production phase in the open shop environment. The scheduling of production and distribution of the products can be done in an integrated way to improve decision-making.

Integration between production and distribution in operations management has become an important factor for operational success in industries. This is because many of the product costs include production and logistics stages throughout the supply chain. It should be mentioned that logistics corresponds to almost 30% of the total costs (MIN; ZHOU, 2002). The strategy for solving these problems in an integrated approach has shown to be better than solving them separately and sequentially (DARVISH; COELHO, 2018).

Therefore, looking for ways to integrate production scheduling and distribution becomes critical for the development of modern companies, integrating scheduling models in which jobs are first processed by machines and then delivered to customers by vehicles.

According to Wang, Grunder and Moudni (2015), integrated scheduling and distribution problems can be classified into two categories: Integrated Scheduling of Production-Distribution Problems (ISPDP) and Integrated Scheduling of Production-Inventory-Distribution Problems (ISPIDP). The former category includes those cases where production and distribution are directly linked, i.e., where the product has to be distributed immediately after production has been finished. This type of scheduling is observed in industries where there are time-sensitive products, such as: newspaper printing and distribution, industrial adhesive material production and delivery and perishable food production and delivery. Examples of these are shown in: (FARAHANI; GRUNOW; GÜNTHER, 2012; BUER; WOODRUFF; OLSON, 1999; DEVAPRIYA; FERRELL; GEISMAR, 2006).

The second category illustrates supply chains where production and distribution are linked through an intermediate inventory aimed at balancing production rates and distribution speed. This integrated model can be found in different supply chains either where production and distribution rates are unbalanced or where there is demand uncertainty, in which the main variations in demand are supplied by the inventory. Many different applications for ISPIDP are described in: (WANG; GRUNDER; MOUDNI, 2015; MOONS *et al.*, 2017).

The objective function of these problems can be classified into two groups: objectives related to costs that involve: setups, Work-in-Process (WIP), costs incurred by maintaining inventory and delivery, and time-related goals, which involve the classical production scheduling indicators such as makespan, tardiness, and earliness (DARVISH; COELHO, 2018). We used the integrated makespan of the system's production and distribution as the proposed problem's objective function. Minimizing the makespan can contribute to improving the service level of production and distribution operations (PINEDO, 2016).

Due to the complexity of the standard open shop scheduling problem, it has become a challenge to develop efficient methods for optimal or sub-optimal problem solutions. No examples of studies were found in the literature that seek to integrate this production environment with product distribution, aiming at assuming more realistic constraints with complex industrial contexts (ANAND; PANNEERSELVAM, 2016; WANG; GRUNDER; MOUDNI, 2015; MOONS *et al.*, 2017; ADAK; AKAN; BULKAN, 2020).

Therefore, this paper aims to develop efficient strategies for solving the open shop scheduling problem with routing by capacitated single vehicle (OSSP-VRP), characterized as an ISPIDP problem minimizing the makespan as an objective function based on time. Four proposals are developed in this work: a Mixed-Integer Linear Programming model (MILP) as an exact solution; a Greedy Insertion Algorithm (GIA) as a constructive heuristic; an Iterated Greedy Algorithm (IGA) a metaheuristic search as an improvement strategy proposed by Tavares-Neto and Nagano (2019); a Biased Random-Key Genetic Algorithm (BRKGA) and a hybrid approach with BRKGA and IGa (BRKGA-IG). To use BRKGA and BRKGA-IG, a new solution decoding/representation scheme will be proposed. Consequently, these methods have never been used in the literature to solve the proposed problem.

There are many other recent algorithms to solve optimization problems using evolutionary techniques (COBAN; CAN, 2010; COBAN, 2013; COBAN; AKSU, 2018; ABREU; PRATA, 2020; LI *et al.*, 2020a; NIU *et al.*, 2021). We consider as benchmarking to optimization methods more two recent competitive metaheuristics. The first is a Cooperation Search Algorithm (CSA) proposed by Feng, Niu and Liu (2021) for numerical optimization and engineering optimization problems, and the second is an Improved Artificial Bee Colony (IABC) algorithm proposed by Li *et al.* (2020b) for combinatorial optimization problems as VRP.

In summary, this paper aims to present a Biased Random-Key Genetic Algorithm with Iterated Greedy Algorithm (BRKGA-IG) for the new problem OSSP-VRP with makespan minimization as the objective function. The main theoretical contributions are: for an efficient generation of the initial solution, a constructive heuristic GIA proposed in this study is used, which produces solutions in competitive computation times. BRKGA-IG has an intensive exploitation mechanism with iterated greedy local search procedure, a restart mechanism to reduce premature convergence of the population, and a new decoding scheme proposed for OSSP-VRP solutions.

To determine the best configuration of the proposed BRKGA-IG method, We used the IRACE package for determining the best execution parameters (LÓPEZ-IBÁNEZ *et al.*, 2016). Extensive computational tests were performed using the well-known OSSP test instances proposed by Guéret and Prins (1998), Taillard (1993) and Brucker *et al.* (1997), adapted for OSSP-VRP. The results show that the developed algorithm outperforms the other procedures tested for the problem. The main experimental contributions are: BRKGA-IG has an average relative percentage deviation smaller than 9% in tested instances, obtained good results in large-sized instances, and BRKGA-IG has competitive computational times compared to other benchmarking methods.

The paper is organized as follows: section 2 provides a literature review of the OSSP and the ISPIDP problems. In section 3, a formal definition of the problem and an example of an instance are illustrated. Section 4 describes the methods adapted from literature for the solution. Section 5 describes the new hybrid method BRKGA-IG. Section 6 performs an extensive analysis of the statistical tests and results. Finally, section 7 discusses the main conclusions of the research and proposes future work for the new problem.

9.2 Literature Review

9.2.1 Open shop scheduling

The Open Shop problem was initially proposed by Gonzalez and Sahni (1976). The problem consists of scheduling a set of jobs on a set of machines, where each processing of a job on a machine (called an operation), has a processing time and can be performed in any sequence. The main objective functions, present in the problem literature, are the minimization of scheduling completion time (makespan), flow time (flowtime) and the sum of tardiness of the jobs (total tardiness) (ANAND; PANNEERSELVAM, 2016).

Regarding the OSSP solution minimizing makespan, several constructive heuristics have been proposed: Pinedo (2016) developed two priority rules based on the bottleneck machine, the Longest Alternate Processing Times (LAPT) and Longest Total Remaining Processing Times on Other Machines (LTRPOM). Liaw (1998) developed the Dense Schedule/Longest Total Remaining Processing (DS/LTRP) dispatch rule as an improvement on LTRPOM. Colak and Agarwal (2005) proposed a heuristic as an amendment to DS/LTRP, creating the Dense Scheduling Longest Total Remaining Processing Time (DS/LTRPAM) rule. Finally, Bai and Tang (2011) proposed a rotation heuristic that achieves good approximate solutions for large problems, the Modified Rotation Scheduling (MRS) heuristic.

Many studies have focused on using metaheuristics to address the topic. For example, Liaw proposed a taboo search (TS) (LIAW, 1999), a simulated annealing (SA) (LIAW, 1999) and a genetic algorithm (GA) (LIAW, 2000) for the problem. The GA performed better than the other methods. Ahmadizar and Farahani (2012) proposed a new hybrid genetic algorithm with a local search strategy that performed better than all other solution strategies already reported. Finally, Hosseinabadi *et al.* (2018) developed an extended genetic algorithm that performed well for large OSSP instances.

The BRKGA Algorithm, hybridized with local search strategies, has shown good results in solving production scheduling problems (CHAVES; GONÇALVES; LORENA, 2018; ANDRADE *et al.*, 2019). No reports were found in the literature regarding the use of BRKGA strategies as metaheuristics and IGA as constructive/improvement heuristics for solving the proposed problem (ANAND; PANNEERSELVAM, 2016). This factor is present in the research, using BRKGA and IGA methods for solving production and distribution scheduling of the OSSP-VRP.

9.2.2 Vehicle routing problem

VRP consists of scheduling one or more vehicles to deliver products from distribution centers to customers. VRP is an extension of the classic traveling salesman problem considering that vehicles have limited capacity and cannot serve all customers in a single route. The objective of the problem is to find, for example, a schedule that minimizes the delivery cost, distance traveled, or routing duration (TOTH; VIGO, 2002).

VRP has many new variants, with applications in the industrial and service sector. Zhang *et al.* (2020) studied the stochastic VRP, considering probability constraints with distribution uncertainty in deadlines. Ghannadpour and Zandiyeh (2020) proposed a multiobjective GA to solve transit vehicle routing problem with vulnerability estimation for risk quantification to optimize the safety of cash/valuable commodities transportation. Finally, Vincent *et al.* (2021) introduced the problem of heterogeneous fleet vehicle routing problem with multiple cross-docks using an adaptive neighborhood simulated annealing algorithm to solve the problem.

Time windows are a critical VRP constraint and have many real applications. This constraint sets pre-defined time intervals that force the vehicles to serve all customers attending these pre-defined time intervals (TOTH; VIGO, 2002). Notable recent contributions are verified for the VRP with time windows. Ali, Côté and Coelho (2020) presented a real-life vehicle routing problem with time windows, synchronization, and heterogeneous fleets. Chen *et al.* (2020) proposed a Multi-Trip Multi-Pickup and Delivery Problem with Time Windows with a customized bus. Molina *et al.* (2020) presented a VRP with time windows and limited number of resources. Finally, Chen, Demir and Huang (2021) studied a VRP with time windows, delivery robots, and a fleet of vans and autonomous robots.

9.2.3 Integrated problems with production and distribution

Concerning the ISPIDP-type integrated problems, considered in this article, the following research can be mentioned: Ullrich (2013) who proposed an environment of parallel machines with distribution with time-window constraints, proposing a genetic algorithm to solve the problem; Farahani, Grunow and Günther (2012) developed a study for perishable products using an MILP model, aiming to improve the quality of service, by optimizing production integrated with the routes. Chang, Li and Chiang (2014) proposed an ant colony optimization (ACO) algorithm to solve the problem of parallel machines integrated with distribution by multiple capacitated vehicles. The results showed the superiority of ACO compared to the exact resolution strategies. Cheng, Leung and Li (2017) developed resolution techniques for the single batch machine problem with delivery by a capacitated single vehicle. For problems with identical processing times, polynomial algorithms were developed for the exact solution. Gao, Qi and Lei (2015) developed heuristic techniques for the general case of the problem studied by Cheng, the techniques, in relation to MILP proved to be promising both in efficiency and speed of execution. Rostami, Nikravesh and Shahin (2018) developed an MILP model for the integrated scheduling of production and distribution of wax with constraints of deterioration of products and learning effect curves.

Concerning the multiple comparison of metaheuristics, Mousavi, Hajiaghaei-Keshteli and Tavakkoli-Moghaddam (2020) developed a SA, GA, particle swarm optimization (PSO) and a hybrid SA strategy with variable neighborhood search (VNS), to solve the integrated production and air transportation scheduling problem with time windows for the due date. They compared different encoding schemes in the proposed solution and their designed algorithms proved to be superior to those already reported in the literature.

Regarding hybridization of ISPIDP with setup times, Tavares-Neto and Nagano (2019) developed an IGA for the parallel machine scheduling problem with sequence dependent setup times, integrating distribution by a capacitated single vehicle. The results obtained showed the superiority of IGA over the exact MILP models.

Concerning the integrated problems with multi-objective functions, Abid, Ayadi and Masmoudi (2020) conducted a study to develop an exact MILP model for the multi-period distribution-planning problem for sea-air intermodal transportation. The model was tested in instances from a real-world case study. Finally, Ganji *et al.* (2020) developed a non-linear integer programming model and a multiobjective GA (NSGA-II) for the multiobjective problem of green production and distribution with heterogeneous fleets and time window constraints, considering objective carbon emission reduction functions. The NSGA II method obtained better results compared to all other tested methods.

The table 35 illustrates the main contributions of the literature to the scheduling problems considering integration between production and distribution. The authors, year of publication, characteristics of the problem such as the type of setup, solution methods and main research contributions are illustrated.

Table 35 – Summary of the main contributions from the literature of scheduling and distribution considering integrated problems.

Author	**Problem characteristics**	**Solution method**	**Contribution**
Farahani, Grunow and Günther (2012)	Perishable products	MILP	The model implementation shows promising results to creates with short time interval between production and delivery.
Ullrich (2013)	Parallel machines	GA	The GA got competitive results with proposed test instances.
Chang, Li and Chiang (2014)	Parallel machines	ACO	ACO was capable of generating near-optimal solutions.
Gao, Qi and Lei (2015)	Parallel machines	Polynomial time algorithms	The algorithms obtain high-quality solution in terms of accuracy and speed with.
Cheng, Leung and Li (2017)	Single batch machine	Polynomial time algorithms	There is a $\mathcal{O}(n \log n)$ algorithm to solve problem concerning jobs with the same size.
Rostami, Nikravesh and Shahin (2018)	Deterioration of products and learning effect curves	MILP	MILP got competitive results in distribution of wax problem.
Tavares-Neto and Nagano (2019)	Parallel machine with setup times	IGA	IGA outperforms GA proposed by Ullrich (2013).
Mousavi, Hajiaghaei-Keshteli and Tavakkoli-Moghaddam (2020)	Air transportation scheduling	SA, GA PSO and VNS	Proposed diferent encoding schemes and the developing methods that outperforms the already reported in the literature.
Abid, Ayadi and Masmoudi (2020)	Sea-air intermodal transportation	MILP	MILP got competitive results in instances from a real-world case study.
Ganji *et al.* (2020)	Green production and distribution	Non-linear model and NSGA-II	The proposed method got solutions that minimizes the carbon emitted by the vehicles with statistical testes to validate results.

Source: Authors.

No examples were found in the literature of using the BRKGA metaheuristic together with the IGA to solve integrated scheduling and distribution problems (WANG; GRUNDER; MOUDNI, 2015; MOONS *et al.*, 2017). Therefore, the paper proposes studying these metaheuristics, comparing their performance with exact strategies, such as MILP and approximates, such as constructive heuristics and single metaheuristics, for the proposed OSSP-VRP problem.

9.3 Problem statement and MILP model

In this section some basic characteristics of the OSSP-VRP are illustrated. The main premises adopted for modelling the production and distribution environment proposed are as follows:

1. The production system considered is the open shop environment. The system comprises a limited set of stages. Each stage has one single machine. The jobs can be processed in different sequences by each of the machines.
2. All jobs are processed in a production environment and stored in a single depot before distribution.
3. There is an intermediate inventory between production and distribution with unlimited capacity for storing jobs with completed production.
4. The distribution system considered is the transportation by a capacitated single vehicle. Each job allocated to a route has a volume; the sum of the volumes of all jobs allocated to a route should comply with the vehicle capacity.
5. The aim of the problem is the minimization of the distribution total time, i.e., the accumulated distribution time of the last job plus the time needed for the vehicle to return to the depot. This is the total length of scheduling (makespan).

The problem consists of scheduling a set of jobs n to be processed on a set of machines m, where each machine and job operation has a processing time. When a job finishes all the operations required for it, the job goes to an intermediate inventory. All jobs are processed and stored in a single depot. Each job is an order, has a volume (size), and must be delivered to a customer by a single vehicle with finite capacity (maximum size). The travel time between the depot and the customers of the jobs, and the customer to another, is defined by a matrix of travel times. Each delivery route must not exceed the vehicle's maximum capacity and must start and end at the depot. The objective of the problem is to minimize the total duration of the scheduling + route planning (makespan), finding a sequence to process the operations and a route to deliver the orders to the customers.

Thus, according to the classification of integrated problems, as proposed by Moons *et al.* (2017), OSSP-VRP is a production scheduling problem in an open shop environment in a single production plant, with intermediate inventory of unlimited capacity and routing by a capacitated single vehicle with limited capacity, multiple trips and deterministic routing times. Since the problem is integration between scheduling and distribution, the classical notation of scheduling problems of Lawler *et al.* (1993) is not applicable.

Tables 36, 37 and 38 show a complete example of an instance for the problem, with three machines and three jobs. Table 36 shows the processing time matrix p_{dj} for each job operation on a machine. In this table, each line corresponds to a processing stage (machine), and each column corresponds to a job. The indices represent machines or jobs (1, 2 and 3 are M_1, M_2 and M_3, respectively or J_1, J_2 and J_3, respectively).

Table 36 – Processing times for open shop example

p_{dj}	1	2	3
1	65	15	20
2	70	50	50
3	30	35	30

Source: Authors.

Table 37 illustrates the travel time matrix σ_{ij} between the depot (location of the plant where the products are produced) and customers. On this matrix, the index 0 represents the depot, and the others represent the customers. The matrix is asymmetric.

Table 37 – Delivery times for open shop example

σ_{ij}	0	1	2	3
0	0	10	20	30
1	30	0	20	10
2	30	10	0	11
3	20	10	11	0

Source: Authors.

Finally, Table 38 shows the volume s_j of each job. The total vehicle capacity Ψ is equal to 10. The total volume allocated to a route must not exceed the capacity Ψ.

Table 38 – Size of jobs (in volume)

	1	2	3
s_j	3	7	5

Source: Authors.

In order to represent this problem in an effective data structure for heuristics modelling, the open shop scheduling to be carried out needs to be represented as the sequence of operation of jobs in machines. Several representation schemes for the solution of the open shop problem have been used in the literature (ANAND; PANNEERSELVAM, 2016; BRÄSEL; TAUTENHAHN; WERNER, 1993). The representation through the list of operations obtained the best performances, when used in heuristic methods (NADERI *et al.*, 2010). The representation is illustrated in Table 39; due to the characteristics of the open shop, operations can be carried out in any sequence.

Table 39 – Operations for the presented instance

Operation	1	2	3	4	5	6	7	8	9
Machine	1	1	1	2	2	2	3	3	3
Job	1	2	3	1	2	3	1	2	3

Source: Authors.

The instance of three jobs and three machines have nine operations (3×3) O_{dj}. Each operation is the processing of job j in machine d. A solution of the open shop scheduling is any sequence of these operations.

The other part of the solution representation is allocating jobs on the routes and their sequence. This structure is represented by an ordered array, where each element corresponds to a route and each route has a sequence of jobs. An example of a solution to the problem described in tables 36 to 38, is illustrated below:

$$S = \{5, 3, 9, 6, 4, 8, 7, 2, 1\}$$

$$R = \{R1 : [3, 1], R2 : [2]\}$$

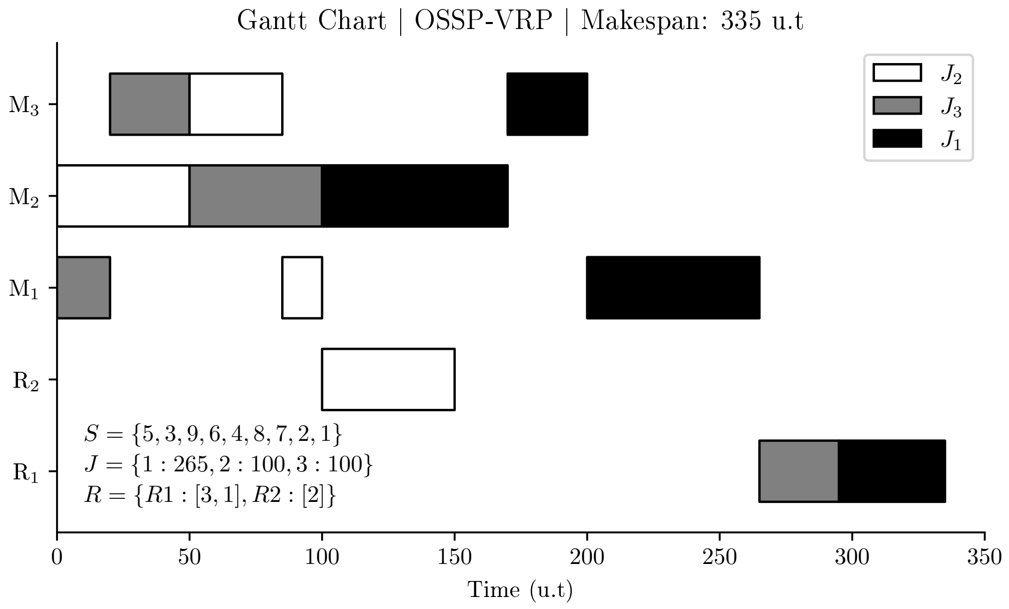
Where S is the sequence of operations to be processed and R is the route sequence. In this case, there are two routes; in the first route, job 3 and job 1 are delivered whereas in the second route only job 2 is delivered.

Figure 68 shows the graphic representation of the problem solution. The Figure illustrates a Gantt chart with the scheduling of jobs and distribution routes. On the y-axis, there are resources such as machines and routes, and on the x-axis, there is time in time units (u.t.).

In the scheduling of the jobs on the machines M, each bar represents the processing time of each job, with the time p_{dj}. The J is the completion time of each job in the production schedule. It is the time that the job is available to enter in a route.

In the scheduling of the R routes, the time of the first job consists of the time spent from depot to customer, whereas the time of the last job of the route consists of the time to arrive at customer and to return to the depot. The time of any intermediate job on the route is the travel time from the previous job to the intermediate job. It can be observed that the routes are ordered by the shortest possible starting time (the starting time of an route is the maximum completion time of all jobs allocated in route).

Figure 68 – Gantt chart for scheduling and routing of solution



Source: Authors

For example, the start time of route R_1 at the depot is 265, then it delivers order 3 and 1 then returns to the depot. The total time would be the start time + σ_{03} + σ_{31} + σ_{10}, ending the delivery at 335 (u.t.).

Therefore, route 1 can only initiate in 265 u.t. time, because of the finishing time of job 1 allocated to the route, route 2 is executed first in the scheduling. Route 2 can be initiated at time of 100 u.t, when job 2 finishes processing. In this solution, there is a total delivery time of 335 u.t. Figure 69 illustrates the optimal solution for the presented instance, with the lowest makespan of 210 u.t.

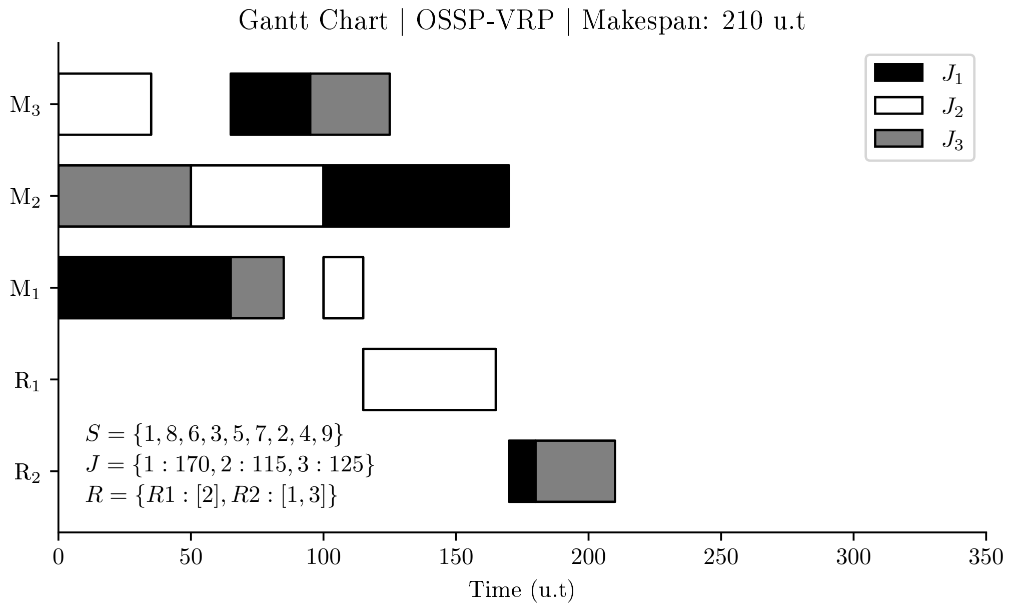
It can be clearly seen that the finishing times of production for jobs 2 and 1 end in sync with the time the routes start. There is no waiting time for these jobs in the intermediate inventory to enter the route and be delivered to the customer.

We proposed the mathematical integer linear programming model in equations (9.1)-(9.27). As the constraints concerning the open shop were created based on a positional notation of the decision variables used to model numerous production programming problems (STAFFORD; TSENG; GUPTA, 2005). The routing and integration constraints were based on the model proposed by Tavares-Neto and Nagano (2019). The sets, parameters and decision variables used to design the mathematical model were defined below. The parameter m is the number of machines and n the number of jobs.

Indices and sets:

d, r: Index used to identify machines {1,2,...,m}.

Figure 69 – Gantt chart for scheduling and routing of best solution



Source: Authors

j, q, l, e: Indexes used to identify jobs {1,2,...,n}.

i, j, h: Indexes used to identify distribution points (*depot* + jobs) {0,1,2,...,n}.

k: Index used to identify a route {1,2,...,n}.

Parameters:

p_{dj}: The processing time of job j in machine d.

s_j: The volume of job j.

Ψ: The total capacity of the vehicle (in volume).

σ_{ij}: Travel time between customer of job i to customers of job j.

M, G e V: Very large constants (used in MILP).

Decision variables:

X_{jdq}: 1, if operation O_{dj} is processed in position q of job j, and 0 otherwise.

Y_{jdl}: 1, if operation O_{dj} is processed in position l of machine d, and 0 otherwise.

CT_{jd}: Completion time of operation O_{dj}.

C_j: Completion time of job j.

w_{ijk}: 1, if job j is delivery by route k immediately after job i, and 0 otherwise.

A_j: Amount of capacity allocated to the vehicle when it arrives at the customer of job j or depot.

R_k: Release time of route k.

D_j: Delivery time of job j at customer.

z: The system makespan.

The mixed-integer linear programming model for the OSSP-VRP is presented below:

minimize

$$z \tag{9.1}$$

subject to

$$CT_{jd} \geq p_{dj}, \qquad \forall d > 0, j > 0 \tag{9.2}$$

$$C_j \geq CT_{jd}, \qquad \forall d > 0, j > 0 \tag{9.3}$$

$$\sum_q X_{jdq} = \sum_l Y_{jdl} = 1, \qquad \forall j > 0, d > 0 \tag{9.4}$$

$$\sum_d X_{jdq} = 1, \qquad \forall j > 0, q \tag{9.5}$$

$$\sum_j Y_{jdl} = 1, \qquad \forall d > 0, l \tag{9.6}$$

$$CT_{jd} \geq CT_{jr} + p_{dj} - (1 - X_{jdq})M - (1 - \sum_{t=1}^{q-1} X_{jrt})M, \qquad \forall j > 0, d > 0, r, q > 1 \tag{9.7}$$

$$CT_{jd} \geq CT_{ed} + p_{dj} - (1 - Y_{jdl})M - (1 - \sum_{t=1}^{l-1} Y_{edt})M, \qquad \forall j > 0, d > 0, e, l > 1 \tag{9.8}$$

$$w_{00k-1} \leq w_{00k}, \qquad \forall k > 1 \tag{9.9}$$

$$\sum_k w_{ijk} \leq 1, \qquad \forall i > 0, j > 0 \tag{9.10}$$

$$\sum_i \sum_k w_{ijk} = 1, \qquad \forall j > 0 \tag{9.11}$$

$$\sum_j w_{0jk} = 1, \qquad \forall k \tag{9.12}$$

$$\sum_j w_{j0k} = 1, \qquad \forall k \tag{9.13}$$

$$w_{jjk} = 0, \qquad \forall j > 0, k \tag{9.14}$$

$$w_{00k} + \sum_{j \geq 0} w_{ijk} \leq 1, \qquad \forall i, k \tag{9.15}$$

$$\sum_{h, h \neq i} w_{ihk} = \sum_{h, h \neq i} w_{hik}, \qquad \forall i, k \tag{9.16}$$

$$A_0 = 0, \tag{9.17}$$

$$s_i \leq A_i \leq \Psi, \qquad \forall i > 0 \tag{9.18}$$

$$A_i \geq A_j + s_i - G(1 - w_{ijk}), \qquad \forall i > 0, j \neq i, k \tag{9.19}$$

$$A_i \leq \sum_{j, j \geq 0} \sum_h (s_j w_{ihk} + G(1 - w_{ihk})), \qquad \forall i > 0, k \tag{9.20}$$

$$R_k \geq C_i - M(1 - w_{ijk}), \qquad \forall i > 0, j, k \tag{9.21}$$

$$D_i \geq R_k + \sigma_{0i} - V(1 - w_{0ik}), \qquad \forall i > 0, j, k \tag{9.22}$$

$$D_j \geq D_i + \sigma_{ij} - V(1 - \sum_k w_{ijk}), \qquad \forall i > 0, j \neq i \tag{9.23}$$

$$R_k \geq D_i + \sigma_{i0} - V(1 - w_{i0k-1}), \qquad \forall i > 0, k > 1 \tag{9.24}$$

$$z \geq D_i + \sigma_{i0} - V(1 - w_{i0k}), \qquad \forall i > 0, k \tag{9.25}$$

$$X_{jik}, Y_{jil}, w_{ijk} \in \{0,1\} \qquad \forall i,j,k,l \qquad (9.26)$$
$$CT_{ij}, C_j, A_j, R_k, D_j, z \in \mathbb{R}^+ \qquad \forall i,j,k \qquad (9.27)$$

The mathematical model is divided into four parts. The first part are the constraints for the open shop problem, with the positional notation, presented constraints (9.2) - (9.8). The second one are the constraints for the single vehicle routing, present in constraints (9.9) - (9.16).

The third part involves the constraints of vehicle capacity and volume of each job, found in constraints (9.17) - (9.20). Finally, the fourth part shows the integration constraints between the open shop scheduling problem and the routing problem, found in equations (9.21) - (9.25). Equations (9.28) - (9.30) illustrate the calculation of the parameters used as sufficiently big numbers in MILP.

Equation (9.2) assumes that the time to complete an operation must be equal or higher than its processing time. Equation (9.3) enforces that the completion time of a job be higher than the completion time of all its operations. Equations (9.4)-(9.6) illustrate the flow control of the open shop decision variables, where each operation has only one position and each position has only one operation. Equation (9.7) and equation (9.8) define the completion time of the operations.

Equation (9.9) ensures that no empty route should exist between two populated routes. An empty route starts and ends in the depot. A populated route starts in the depot, visits customers, and ends in the depot. Equation (9.10) indicates that each arc (i,j) can be part of only one route. Equation (9.11) indicates that only one route visits a customer. Equation (9.12) and Equation (9.13) ensures that each route enters and leaves the depot only once. Equation (9.14) prohibits the vehicle from entering and leaving the same customer. Equation (9.15) enforces that route k does not enter and leave the depot, when it is used in the solution. Equation (9.16) keeps the flow control of the routing.

Equation (9.17) sets the capacity allocated to the vehicle when it arrives at the depot (end of a route) must be equal to zero, the depot has index 0. Equation (9.18) and Equation (9.19) establish the lowest and highest load limits when it arrives at customers. Equation (9.20) and Equation (9.21) establish the removal of sub-routes, based on the MTZ constraints (DESROCHERS; LAPORTE, 1991). In addition, Equation (9.21) indicates that the release time of a k route must be higher than the completion time of the jobs allocated to that route.

Equation (9.22) and Equation (9.23) calculate the job delivery time to the customer. Equation (9.24) enforces that a route´s starting time is higher than the finishing time of the previous route. Equation (9.25) calculates the objective function value as the time of completion of the last route added to the vehicle´s returning time to the depot. Finally, Equation (9.26) and Equation (9.27) describe the domain of the decision variables.

$$M = \sum_{d} \sum_{j} p_{dj} \tag{9.28}$$

$$G = \sum_{i} s_i \tag{9.29}$$

$$V = M + \sum_{i} \sum_{j} \sigma_{ij} \tag{9.30}$$

Equations (9.28)-(9.30) show the calculation of the Big-M parameters for modeling the "if-then" constraints. The MILP model proposed has altogether $nm\,(n+m) + n\,(n+1)^2 + n\,(m+4) + 3$ decision variables, where $nm\,(n+m) + n\,(n+1)^2$ are binary variables and $n\,(m+4) + 3$ are real variables. The model has $5nm + nm^2\,(m-1) + n^2m\,(n-1) + n\,(3n^2+9n+7)$ linear constraints. The constraint with the largest size is the one found in equations (9.7) and (9.7) with the worst case complexity of $\mathcal{O}\,(m^2n^2)$. The Table 40 illustrates a comparison of number of constraints and decision variables of MILP, for several different instance sizes.

Table 40 – Comparison of MILP formulation with examples of instances sizes for OSSP-VRP

Instances sets		MILP		
m	n	# integer variables	# continuous variables	# constraints
3	3	102	24	336
4	4	228	35	828
5	5	430	48	1760
6	6	726	63	3354
7	7	1134	80	5880
8	8	1672	99	9656
9	9	2358	120	15048
10	10	3210	143	22470
15	15	10590	288	107880
20	20	24820	483	333740

Source: Authors.

9.4 Optimization methods

As the problem is a generalization of the classical open shop, there is a high number of viable solutions, which is considered a problem hard to solve (GUÉRET; PRINS, 1998). Exact approaches for this problem are hard to apply in practical situations with large-scale instances, due to the high computational costs incurred. Therefore, heuristic and metaheuristic methods need to be used. Thus, the proposal is: a greedy insertion heuristic, based on the NEH constructive technique (NAWAZ; JR; HAM, 1983), as a heuristic method and an iterated greedy algorithm and a biased random key genetic algorithm, as metaheuristic methods. Moreover, the proposal also includes a hybrid approach joining together the three resolution strategies considered.

The following subsections describe the GIA, IGA, and BRKGA methods adapted for OSSP-VRP. Section 5 illustrates the new proposed hybrid method BRKGA-IG that uses a hybridization of BRKGA and IGA to improve the solutions' quality.

9.4.1 Greedy insertion algorithm (GIA)

The proposed insertion heuristic is an adaptation of the insert-and-order algorithm proposed by Tavares-Neto and Nagano (2019). In each iteration, the heuristic is based on two steps: the first is the ordering of operations and the second is the insertion of operations, in all possible positions in the operations sequence and the insertion of jobs, in all possible positions in the routes, returning the insertion of the operations and routes that obtain the best makespan.

GIA is based on the NEH constructive heuristic, adapted for the OSSP-VRP. The operations are tested in all possible positions of the operations vector, whereas the jobs are tested in each route and the best position considered in each route is the one that has the cheapest insertion cost, as in Equation (9.31), where Δ is the insertion cost of job j between job i and job e in the route.

$$\Delta = \sigma_{ij} + \sigma_{je} - \sigma_{ie} \tag{9.31}$$

Figure 70 illustrates the pseudo-code of GIA; the algorithm receives the V list as input with the ordered operations in decreasing order of processing time. The algorithm returns the S^* operations vector and the R^* route vector obtained with the best insertions carried out. The heuristic proposed is not dependent on the parameters.

```
   Data: $p$, $\sigma$, $s$, $\Psi$
   Result: A operation sequence $S^*$ and route $R^*$
1  $V \leftarrow$ set of operations sorting in non-increasing order of processing time $p$;
2  $S$, $S^* \leftarrow \{\}, \{\}$ operation sequences;
3  $R$, $R^* \leftarrow \{\}, \{\}$ route sequences;
4  foreach $\pi_{dj} \in V$ do
5      foreach position $q \in S^*$ do
6          $S \leftarrow$ solution with $\pi_{dj}$ inserted in position $q$ in $S^*$;
7          if job $j$ of operation $\pi_{dj}$ was processed totally then
8              foreach route $r \in R^*$ do
9                  $R \leftarrow$ solution with insertion of job $j$ in route $r$ in position with best
                    saving $\Delta = \sigma_{ij} + \sigma_{je} - \sigma_{ie}$ respecting the total volume $\Psi$;
10                 compute makespan of new $S$ and $R$ solutions;
11             end
12         end
13     end
14     Update $S^*$, $R^*$ with the best solution found;
15 end
```

Figure 70 – The pseudo-code for the GIA heuristic

Lines 1-3 lines initiate the data structures of the algorithm. In line 4, the main GIA loop is initiated, where each operation π_{dj} is tested in every possible position in line 6. Line 7 tests if all the operations related with j job were inserted, if true, the algorithm tests the insertion of job j in all routes, and the position that obtains the smallest saving is selected. This test is performed in line 9. Finally, in line 14 solutions S^* and R^* are updated with the best insertions found in the operations and routes vector, respectively.

9.4.2 Iterated greedy algorithm (IGA)

The proposed iterated greedy algorithm is an adaptation of the IGA algorithm proposed by Tavares-Neto and Nagano (2019), for the OSSP-VRP. The IGA algorithm proposed works as a GIA constructive heuristic natural extension. The IGA is an algorithm widely used for scheduling problems, such as flow shop (RUIZ; STÜTZLE, 2008; JIN; SONG; WU, 2007).

The IGA is a type of stochastic local search, in which its basic framework consists of three main processes:

Generation of the initial solution: This is the structure that provides the initial solution for the algorithm to be executed from it. In this study, it is the solution generated by the GIA algorithm.

Destruction phase: One part of the current solutions is removed using a given criteria. In the study, a percentage of the operations are randomly removed from the solution.

In the case in which an operation of a job is removed, that job is also removed from the route to which it was allocated.

Construction phase: A structure is used that builds a new solution from the partial solution obtained from the destruction phase. In the proposed algorithm, the GIA algorithm is executed again from the partial solution of the operations and routes vector.

Figure 71 illustrates the IGA pseudo-code; the algorithm receives two parameters as input: *Ncycles* is the number of cycles (iterations) of IGA and that *pr* is the percentage of operations to be removed randomly in the destruction phase. Those two parameters are dependent on the size of the problem and must be optimized. The algorithm returns the best S^* operation vector and the best vector with the routes R^* obtained among all iterations of the algorithm performed.

```
   Data: $p$, $\sigma$, $s$, $\Psi$
   Result: A operation sequence $S^*_\pi$ and route $R^*_\pi$
 1 $Ncycles \leftarrow$ number of iterations;
 2 $pr \leftarrow$ percentage of operations removed in the destruction phase;
 3 $S_\pi, R_\pi \leftarrow GIA(, \Psi)$;
 4 $S^*_\pi \leftarrow S_\pi$;
 5 $R^*_\pi \leftarrow R_\pi$;
 6 for $i = 1$ to $Ncycles$ do
 7     $S'_\pi, R'_\pi \leftarrow Destruction(S_\pi, R_\pi, pr)$;
 8     $S_\pi, R_\pi \leftarrow GIA(S'_\pi, R'_\pi)$ // Construction phase
 9     if $makespan(S_\pi, R_\pi) < makespan(S^*_\pi, R^*_\pi)$ then
10         $S^*_\pi \leftarrow S_\pi$;
11         $R^*_\pi \leftarrow R_\pi$;
12     else
13         $S_\pi \leftarrow S^*_\pi$;
14         $R_\pi \leftarrow R^*_\pi$;
15     end
16 end
```

Figure 71 – A generic pseudo-code for IGA

Lines 1-5 initiate the parameters and data structures of the algorithm. In line 6, the main IGA loop initiates, where each destruction and reconstruction pair is executed. In line 7, the algorithm destruction phase is executed, where $N \times pr$ operations are removed randomly from vector S_π, where N is the size of the problem. For all removed operations, their respective jobs in route R_π are removed.

In line 8, the IGA construction phase is carried out; the GIA algorithm is executed from the partial solution S'_π, R'_π. The algorithm is executed testing the insertions of the operations insertions not yet inserted in the solution. In lines 9-14, if the new solution obtained is better than the best solution found, the best solution is then updated and the construction phase starts again. Otherwise, the algorithm returns to the construction phase still with the best current solution. It is clear that the acceptance criterion is of the hillclimbing type (RUIZ; STÜTZLE, 2008).

9.4.3 Biased random key genetic algorithm (BRKGA)

The BRKGA algorithm is a populational metaheuristic, based on the key concepts of the genetic algorithm, with each individual of the population as a vector with random keys (GONÇALVES; RESENDE, 2011), where the crossover between individuals occurs in a biased manner, favouring the genetic material from the population‘s best solutions. The BRKGA was initially designed to solve permutational problems, where the solution could be structured as a single-value sequence (ANDRADE *et al.*, 2019).

Because of this structure, the BRKGA has been widely used to solve production scheduling problems because usually the solution of these problems can be represented as a single-value sequence, as a list of the sequence of jobs or operations processing, in the OSSP case.

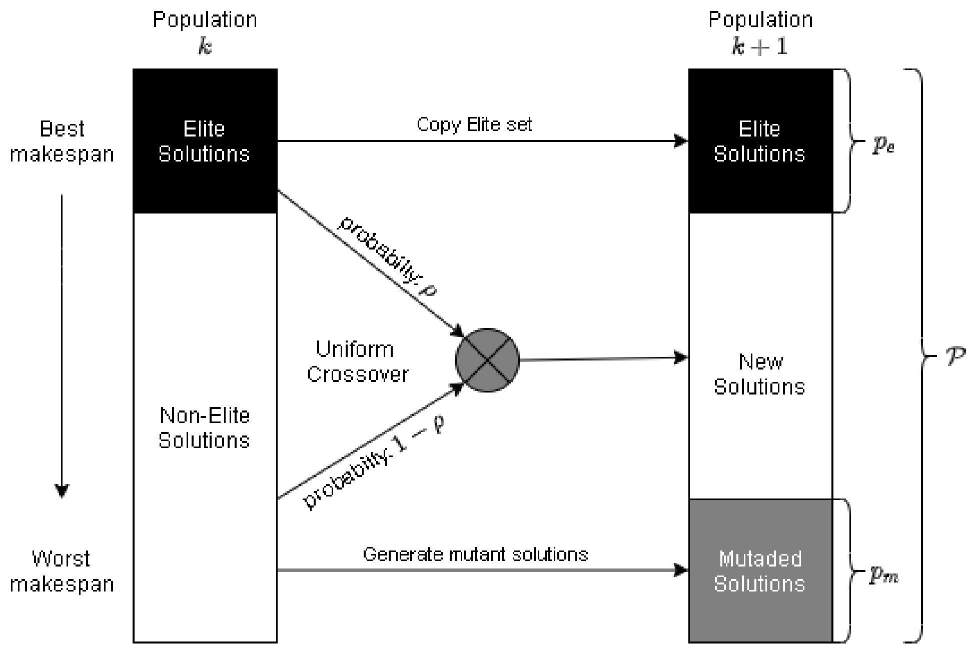
The BRKGA performed better compared to the classical metaheuristics proposals for industrial engineering problems, as in: Gonçalves and Resende (2004) that developed a BRKGA for the cell formation problem, which performed better than classic GA. The BRKGA gets good performance for job shop problems, as in (GONÇALVES; MENDES; RESENDE, 2005) and (BEIRÃO *et al.*, 1997). Obtaining better results than the metaheuristics SA and Tabu Search.

Andrade, Silva and Pessoa (2019) developed a BRKGA for a flow shop problem minimizing the total flowtime, proposing a new perturbation mechanism of the solution, population reset and local search. The algorithm obtained better results than an ILS and IGA. Finally, Andrade *et al.* (2019) developed a BRKGA with a path relinking structure as a local search. They obtained a better performance than the classic BRKGA strategies for the firmware-over-the-air scheduling problem (FOTA).

The proposed BRKGA algorithm starts by generating p random keys with values between 0 and 1. Two individuals from this population are replaced by solutions generated by the GIA algorithm, the first with the vector of operations V increasing order of processing times and the second in decreasing order, as a warm-start strategy. The individuals generated by the GIA, are encoded in the form of random keys to be inserted into the population.

After having created the initial population, the BRKGA initiates its cycle of generations. Figure 72 illustrates the process of a generation of the algorithm; it is used a decoding function that converts the random key in the value of the makespan of solution. Therefore, all the solutions are ordered in relation to the makespan. The best p_e are considered from the elite set and added directly to the new population.

Figure 72 – BRKGA scheme for population evolution



Source: Authors

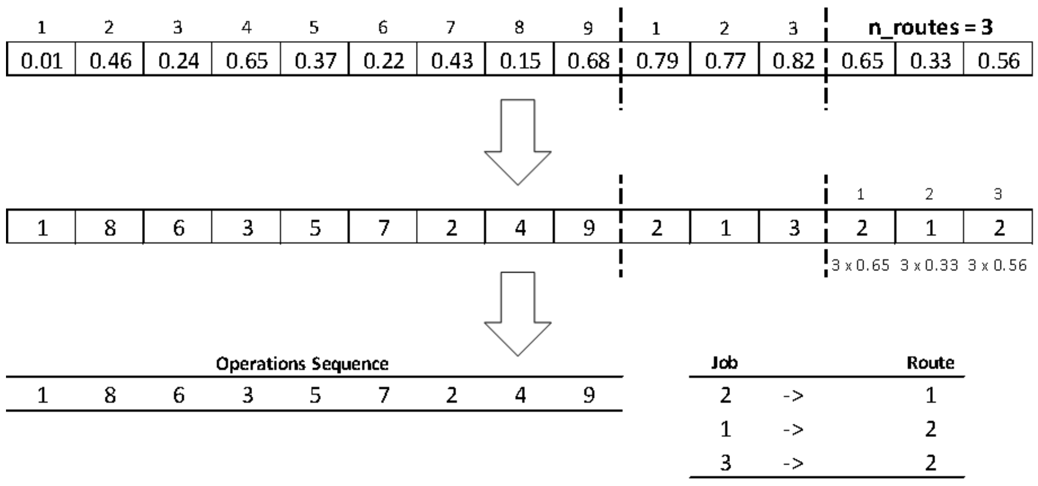
The worst p_m solutions are replaced by the new mutating solutions, created according to the initial population generation procedure. The rest of the solutions from the new population are inserted through a $p - p_e - p_m$ crossover execution between an elite group member and a non-elite group member. A uniform crossover operator is used, the probability of the father elite gene insertion is ρ. These steps are executed by a certain number of generations or until a time limit. The BRKGA has the following parameters: p_e, p_m, ρ, p, *num_gen* and *time_limit*, these parameters need to be optimized.

The only BRKGA operator that is dependent on the problem is the decoding operator. It must be formulated so that it is possible to convert the characteristics of the solution in the form of random keys for the problem domain, so that it is possible to calculate the objective function. Figure 73 illustrates the proposed decoding scheme.

An example of a random key is illustrated for the best solution of the example problem, shown in figure 69. The key has a size of $m \times n + n + n$. Each dashed line represents a block of the solution. Where the first block represents the sequence of operations, the second the sequence of jobs in routes and the third the allocation of jobs on each route.

The jobs in routes and operations sequences are obtained by ordering the vector in its respective blocks with increasing order. Therefore, the jobs in routes or operations are the indexes of vectors, ordering by values of each index in increasing order. The assignment

Figure 73 – Proposed decoder for OSSP-VRP



Source: Authors

of the jobs to the routes is obtained by multiplying the value of each key, with index related to a job, by the total number of routes, which is equivalent to the number of jobs of the instance, rounding the value up. The sequence of routes are ordered by the shortest possible starting time. The starting time of the route is the maximum production completion time among all jobs allocated in route.

With this decoding scheme unfeasible solutions can still be obtained, with routes that do not comply with the capacity Ψ of the vehicle. Therefore, a penalty mechanism of the solution is proposed. Equation (9.32) describes the calculation of the decoded solution fitness, applying the penalty operator.

$$Fitness = makespan + \sum_{r \in Routes | v_r > \Psi} (v_r - \Psi) \times \Psi \tag{9.32}$$

Where v_r is the amount of volume assigned to route r and Ψ is the capacity of the vehicle. Thus, if there is a route in which the capacity of the vehicle is exceeded, a penalty is added to the makespan, which is proportional to the exceding volume in the route.

9.5 The new hybrid approach (BRKGA-IG)

The proposed hybrid approach uses local search strategies in metaheuristics that provoke considerable improvements in algorithms, intensifying the exploitation phase of the neighborhood of solutions (ANDRADE; SILVA; PESSOA, 2019). Therefore, the present paper proposes the hybridization of BRKGA and IGA, with IGA as a local search strategy (BRKGA-IG). The BRKGA-IG uses IGA as a search algorithm in the exploitation phase to improve solutions during execution of generations. Also, is added a restart procedure to reduce premature convergence of the population, where to each R iterations without improvement, the population is restarted, keeping only the best solution.

The figure 74 illustrates the complete pseudo-code of BRKGA-IG. The algorithm is dependent on the following parameters: p_e, p_m, ρ, p, L, R, *gen* e *time_limit*.

```
   Data: $p$, $\sigma$, $s$, , $\Psi$, $p_e$, $p_m$, $\rho$, $p$, $L$, $R$, $gen$ e $time\_limit$
   Result: A $bestsol$ solution found
1  $p \leftarrow$ number of individuals in population $\mathcal{P}$;
2  $p_e \leftarrow$ percentage of elite set;
3  $p_m \leftarrow$ percentage of mutant set;
4  $\rho \leftarrow$ probability to select a elite father key in crossover;
5  $L \leftarrow$ number of generations to apply IGA;
6  $R \leftarrow$ number of generation without improvement to apply restart;
7  $gen \leftarrow$ number of generation;
8  Initialize population $\mathcal{P}$ with $p$ random individuals and GIA solutions;
9  $bestsol \leftarrow$ best GIA solution found;
10 $k \leftarrow 0$ ;                    // Number of generations without improvement
11 $iter \leftarrow 1$;
12 while $iter < gen$ or $run\ time\ does\ not\ end$ do
13     Evolve $\mathcal{P}$ one generation ;          // Using $\mathcal{P}$, $p$, $\rho$, $p_e$ and $p_m$
14     $sol \leftarrow$ best solution $\in \mathcal{P}$;
15     if $iter \mod L = 0$ then
16         Apply IGA in $sol$ and inject chromosome in population $\mathcal{P}$;
17     end
18     if $makespan(sol) \geq makespan(bestsol)$ then
19         $k \leftarrow k + 1$ ;             // The best solution was no improvement
20         if $k \geq R$ then
21             Replace population $\mathcal{P}$ with $bestsol$, GIA solutions and random
                 individuals;
22         end
23     else
24         $k \leftarrow 0$;
25         $bestsol \leftarrow sol$;
26     end
27     $iter \leftarrow iter + 1$
28 end
29 return $bestsol$
```

Figure 74 – A generic pseudo-code for a BRKGA-IG

In lines 1-11, starts the parameters and data structures of BRKGA-IG. In line 12, there is the main while loop of the algorithm. The algorithm runs while the number of generations or the execution time limit is not exceeded. In line 13, the generation process of BRKGA is executed according to figure 72. In lines 14-17, the algorithm applies to each L generations the IGA procedure in the best solution obtained in that generation.

In lines 18-26, the BRKGA-IG tests if the generation found no better solution than *bestsol*. If the number of generations without improvement is greater than R limit, the algorithm executes the restart in line 21. Otherwise, the best solution is updated and is reset the k counter. In line 29, the BRKGA-IG returns the best solution found.

9.6 Results and analysis

The exact and approximate optimization methods were evaluated using literature benchmarking problems. These problems are instances proposed by Guéret and Prins (1998), Taillard (1993) and Brucker *et al.* (1997). These instances are a set of data usually used for testing algorithms to solve classic OSSP. We adapted these instances to the OSSP-VRP problem.

In Guéret and Prins (1998) instances, the processing times are random values, uniformly distributed between 1 and 1000. The problem sizes considered were: $n, m \in \{3, 4, 5, 6, 7, 8, 9, 10\}$. There are 10 instances for each problem class, totalizing 80. In Taillard (1993) instances, the processing times are random values, uniformly distributed between 1 and 100. The problem sizes considered were: $n, m \in \{4, 5, 7, 10, 15, 20\}$. There are 10 instances for each problem class, totalizing 60. In Brucker *et al.* (1997) instances, the processing times are random values uniformly distributed between 1 and 500. The problem sizes considered were: $n, m \in \{3, 4, 5, 6, 7, 8\}$, totalizing 60 instances. In summary, we adapted 192 instances to OSSP-VRP.

The parameters s_j, σ_{ij} and Ψ were randomly created, based on the procedure proposed by Tavares-Neto and Nagano (2019). The volume of order s_j, is an integer between $[1, 10]$. The travel time, σ_{ij} is determined by generating points in a plane $\theta_s \times \theta_s$, where θ_s is an integer between $\{10, 20, 30\}$, the travel time is calculated based on the euclidean distance between the points. Finally, the vehicle capacity Ψ is a random integer between $[\max(\sigma), 5 \times \max(\sigma)]$.

We implemented all the proposed and benchmarking algorithms in the Julia 1.5.1 language (https://julialang.org/). The MILP model was implemented in the IBM ILOG CPLEX 12.10 solver. The computational experiments were executed in a virtual machine with Intel® Xeon(R) CPU E5-2660 2.20GHz and 12GB of RAM. All the source codes, instances sets, and results of the algorithms are available at http://repositorio.uspdigital.usp.br/handle/item/443.

To obtain the best parameters of each proposed metaheuristic, the IRACE package (LÓPEZ-IBÁNEZ *et al.*, 2016) was used. The table 41 illustrates the parameters considered for tuning each of the algorithms and the best parameter obtained in the IRACE tests, where $N = m \times n$ is the problem's size. The IRACE package is used to find the best combination of parameters for optimization algorithms; IRACE provide the information of the test results for a set of combination parameters. For example, in table 41 for the BRKGA algorithm, IRACE recommends adopting intermediate values of the parameters

p_e, p_m, ρ and gen and the smallest available value for parameter p equal to $10 \times N$.

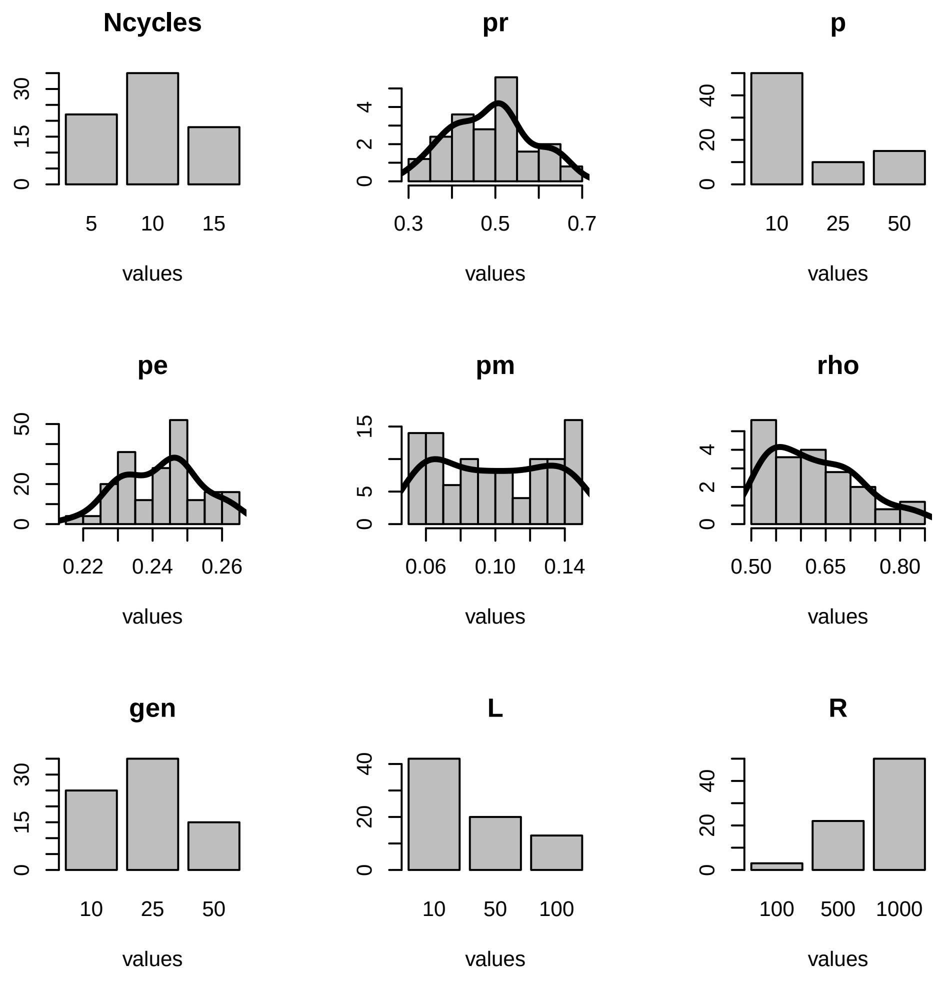
Table 41 – IRACE parameter range settings and resulting values

Parameters	IGA		BRKGA					BRKGA-IG	
	$Ncycles$	p_r	p	p_e	p_m	ρ	gen	L	R
Proposed	$\{5, 10, 15\}$	$[0.30, 0.70]$	$\{10, 25, 50\}$	$[0.15, 0.30]$	$[0.05, 0.15]$	$[0.50, 0.85]$	$\{10, 25, 50\}$	$\{10, 50, 100\}$	$\{100, 500, 1000\}$
IRACE	$10 \times N$	0.55	$10 \times N$	0.25	0.15	0.55	$25 \times N$	10	1000

Source: Authors.

During the IRACE execution, the algorithm interactively updates the form of parameter sampling, aiming to use the best parameters region that improves the performance of the optimization algorithms. The sampling frequency of the parameters can provide important insights into the behavior of the optimization algorithm when the parameters are changed (LÓPEZ-IBÁNEZ *et al.*, 2016). Figure 75 shows the parameters sampling frequency for developed methods.

Figure 75 – Parameters sampling frequency for developed methods



Source: Authors

The parameters related to the generations of BRKGA-IG were the same used in BRKGA. For the BRKGA and BRKGA-IG algorithms were defined 30 min of execution time, as one more stop criterion, in addition to the number of generations. The maximum execution time of MILP was the same as other approximation methods (30 min).

For comparison purposes, the parameters for the number of generations and maximum execution time of the CSA and IABC algorithms were the same as those used in the BRKGA and BRKGA-IG algorithms. We set the other CSA parameters according to Feng, Niu and Liu (2021). The IGA was used as a global search in the IABC algorithm, with the other parameters set according to Li *et al.* (2020b).

The indicator used to measure the efficacy of the computational experiments is the gap between the solution obtained by the method (sol_{ik}) and the best-known solution for the instance of the classic OSSP (BKS_i), as an estimate of the lower bound for the OSSP-VRP. Therefore, it may be reduced to a classic open shop problem, eliminating the routing parameters. The equation (9.33) illustrates the relative percentage deviation indicator, where sol_{ik} is the value of the instance i obtained by the method k and BKS_i is the best solution of the classic OSSP, for instance i.

$$RPD_{ik} = \frac{sol_{ik} - BKS_i}{BKS_i} \cdot 100 \tag{9.33}$$

In summary, the methods considered for the comparison between performance and computational times were:

- Mixed-Integer Linear Programming (MILP): the proposed MILP model.
- Greedy Insertion Algorithm (GIA): a constructive heuristic developed by us.
- Cooperation Search Algorithm (CSA): a competitive metaheuristic proposed by Feng, Niu and Liu (2021).
- Improved Artificial Bee Colony (IABC): a competitive metaheuristic proposed by Li *et al.* (2020b).
- Iterated Greedy Algorithm (IGA): a meta-heuristic adapted for OSSP-VRP and initially proposed by Tavares-Neto and Nagano (2019).
- BRKGA with a new decoding scheme (BRKGA): a meta-heuristic proposed by us.
- BRKGA hybridized with IGA (BRKGA-IG): a meta-heuristic proposed by us.

Table 42 illustrates the average RPD (ARPD) for each set of instances and the optimization methods. In bold are the best results obtained for each set of instances. In the last three lines, there is a statistical summary of the optimization methods tested.

Table 42 – Results of optimization methods for each set of instances

Group	Size	MILP	GIA	CSA	IABC	IGA	BRKGA	BRKGA-IG
Gueret and Prins	3	**1.37**	22.95	**1.37**	**1.37**	**1.37**	**1.37**	**1.37**
	4	**1.48**	12.94	1.77	1.59	1.53	**1.48**	**1.48**
	5	2.61	24.59	1.92	1.77	1.30	1.16	**1.15**
	6	6.24	33.75	3.61	1.90	1.31	1.20	**1.04**
	7	29.90	27.46	5.74	4.78	2.60	1.93	**1.13**
	8	46.00	49.00	13.15	10.42	8.45	8.22	**5.14**
	9	41.41	52.46	13.14	12.69	11.08	9.49	**6.33**
	10	45.97	51.32	16.79	16.23	16.36	12.79	**9.36**
Taillard	4	**15.39**	44.48	17.53	16.77	15.42	18.02	**15.39**
	5	16.80	40.90	17.08	15.94	15.67	18.18	**14.70**
	7	39.47	47.05	14.67	14.21	16.45	13.40	**12.10**
	10	50.29	41.47	15.34	13.44	20.40	45.99	**11.58**
	15	50.42	37.69	16.14	19.18	18.79	72.05	**15.05**
	20	62.40	39.20	20.13	23.34	21.53	28.38	**18.34**
Brucker	3	**1.71**	10.59	**1.71**	**1.71**	**1.71**	**1.71**	**1.71**
	4	**1.88**	30.56	3.45	4.08	1.96	2.22	**1.88**
	5	5.48	38.87	5.75	5.54	4.63	4.59	**3.30**
	6	23.73	45.66	7.45	6.65	8.08	6.23	**4.14**
	7	47.86	51.64	11.27	11.53	15.16	10.11	**7.75**
	8	48.94	58.42	14.42	13.20	19.36	15.39	**10.13**
	Min	1.37	10.59	1.37	1.37	1.30	1.16	**1.04**
	Average	26.97	38.05	10.12	9.82	10.16	13.70	**7.15**
	Max	62.40	58.42	20.13	23.34	21.53	72.05	**18.34**

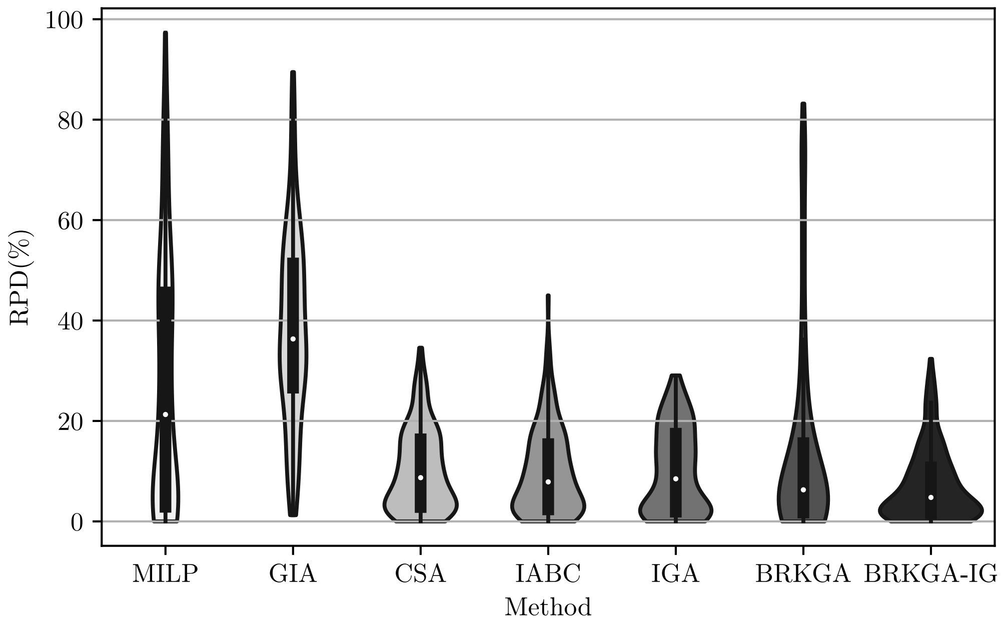
Source: Authors.

Regarding the results of the instances of Gueret and Prins, most of the methods obtained good results in instances of size 3 and 4. For the larger instances, BRKGA-IG got the best results. For Taillard instances, the ARPD results, in general, presents high values, with the BRKGA-IG metaheuristic obtaining the best results in instances of size greater than 4. Finally, for Brucker instances, the IGA and BRKGA-IG obtained similar results, with a greater difference in instances with sizes between 7 and 8.

Figure 76 shows a violin graph of the RPD distribution obtained by the optimization methods in all tested instances. In general, BRKGA-IG obtained the best results, followed by IGA and BRKGA. The CSA e IABC got similar results of RPD. The GIA heuristics got the worst results, denoting the difficulty of solving the problem using simple heuristic methods.

Concerning the RPD distribution, in figure 76, the BRKGA-IG method has the lowest median in comparison to the other methods, although it is very similar to the

Figure 76 – RPD for all optimization methods in all sets of instances



Source: Authors

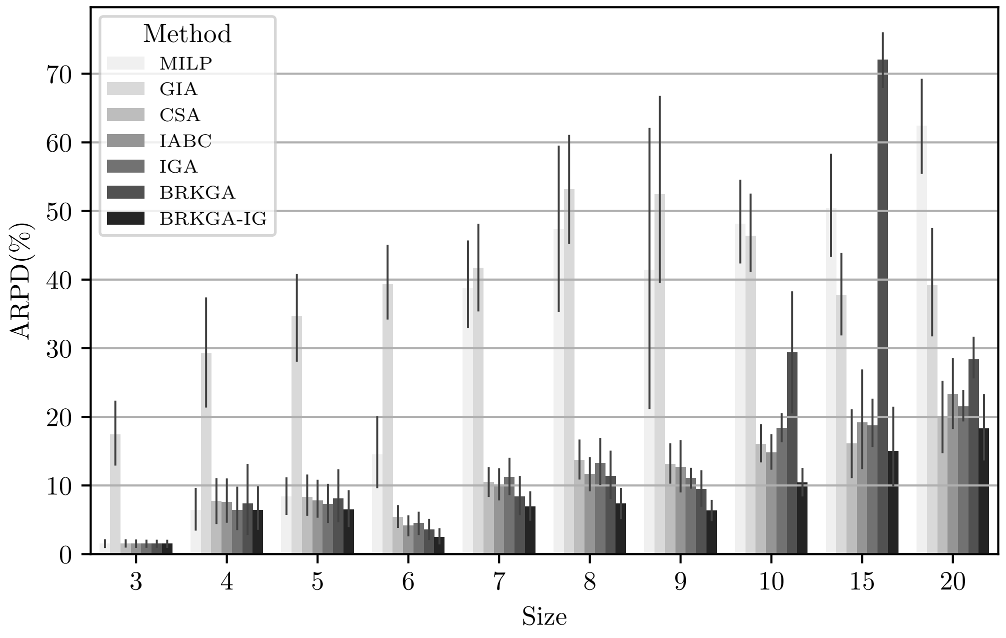
results of the CSA, IABC and IGA median. However, the RPD distribution of BRKGA-IG has more values closer to zero than other methods because the base of the distribution is denser. For a better comparison, figure 77 illustrates the ARPD of each optimization methods, grouped by instance size. The figure illustrates the comparison between our proposed algorithms (MILP, GIA and BRKGA-IG) and other benchmarking algorithms (CSA, IABC, BRKGA and IGA).

Concerning figure 77, it is possible to verify that the MILP obtains a good performance up to the instances of size 6, with the loss of performance increasing gradually with the increase of instance size. It is indicating the difficulty of solving the complex problem by exact methods. The constructive heuristics GIA gets the worst performances in relation to the others ranging in GAPs between 20% and 50%. In larger instances, GIA gets better performance than MILP.

Among the metaheuristic methods, they have a similar performance until the instances of size 9. BRKGA obtains the worst results, especially in size 15 instances, and this may indicate that the generation of large-size instances of Taillard (1993) is difficult to solve by the method. In larger instances, BRKGA-IG obtains better performance than CSA, IABC, BRKGA and IGA, meaning that with the hybridization of the methods, it is possible to improve OSSP-VRP solutions, especially with larger instances. The GAPs of the instances sizes greater than 9 are, on average, smaller than 20%, showing that BRKGA-IG is promising to solve large-size instances.

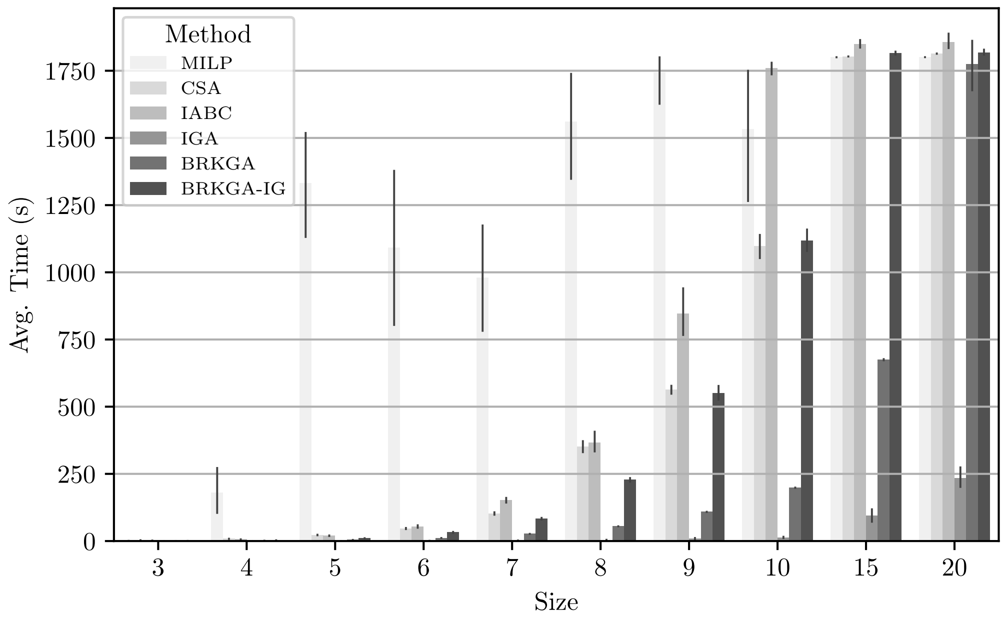
Regarding computational times, figure 78 illustrates the average computational

Figure 77 – ARPD and confidence interval ($\alpha = 0.05$) for all optimization methods in each instance size



Source: Authors

Figure 78 – Average computational times and confidence interval ($\alpha = 0.05$) for all optimization methods in each instance size



Source: Authors

times obtained in each method for each grouped instance size. To analyze the computational time with increasing instance size, table 43 shows average computational times of all optimization methods for each set of instances. The GIA heuristics presents computational times smaller than 1 second, so it was not considered in the visualizations.

Table 43 – Average computational times of optimization methods for each set of instances

Group	Size	MILP	CSA	IABC	IGA	BRKGA	BRKGA-IG
Gueret and Prins	3	0.71	2.56	1.62	0.09	0.37	0.61
	4	417.79	7.27	5.36	0.06	1.36	2.77
	5	1801.40	19.32	16.91	0.24	4.25	10.75
	6	1441.25	43.13	46.63	0.83	11.42	31.16
	7	986.12	86.77	125.21	1.96	26.83	88.77
	8	1759.28	313.35	313.92	3.43	54.96	228.97
	9	1742.39	563.47	845.11	8.84	108.52	550.24
	10	1800.03	1030.41	1730.10	14.51	198.27	1170.60
Taillard	4	30.94	9.21	6.81	0.06	1.32	2.73
	5	1206.73	25.09	22.37	0.19	4.22	9.94
	7	1187.10	110.87	169.55	1.43	26.67	79.88
	10	1265.32	1164.50	1789.83	12.27	198.59	1064.79
	15	1800.28	1803.21	1849.71	94.78	674.80	1815.47
	20	1800.55	1813.69	1856.63	234.82	1774.79	1817.14
Brucker	3	0.70	3.05	1.84	0.01	0.33	0.51
	4	81.99	8.43	6.43	0.07	1.33	2.69
	5	945.57	23.32	20.20	0.22	4.24	10.38
	6	704.89	50.43	61.39	0.77	11.36	33.67
	7	743.09	110.94	162.82	1.98	26.60	80.12
	8	1309.92	396.35	431.21	4.81	54.85	225.94
	Min	0.70	2.56	1.62	0.01	0.33	0.51
	Average	1051.30	379.27	473.18	19.07	159.25	361.36
	Max	1801.40	1813.69	1856.63	234.82	1774.79	1817.14

Source: Authors.

It is perceptible that the MILP model with instances size greater than 7 presents expressive computational times, indicating the difficulty of solving larger problems by exact methods. The IGA presents the best computational times because it is a single solution-based metaheuristic. The BRKGA-IG, as a population-based metaheuristic, gives the highest computational costs. The CSA and IABC presents similar computational cost until instances with size 8. For larger instances, the IABC present high computational times than CSA and BRKGA-IG. The MILP presents the most considerable variability between the computational times.

For a fair comparison, all metaheuristic methods used the same criteria of computational time and number of iterations (LATORRE *et al.*, 2020). With these criteria, the proposed method BRKGA-IG outperformed the other methods tested in solution quality. For the same number of iterations, 10 x N and execution time limit of 30 min BRKGA-IG obtained better RPD results. MILP is an exact method with time-dependent branch and bound interactions, GIA is a deterministic method, so it has a small duration with a time less than 1 second. The other metaheuristics have the same basis of comparison and can stop at local optimum, which indicates that the solution will not improve if more computational time is provided. The proposed method has a restart scheme to escape convergence to a local optimum.

To validate the results, it is essential to verify if the ARPD differences are statistically significant among better performance methods. It was then applied the t-test for two samples (MONTGOMERY, 2017) between the BRKGA-IG and CSA methods, with $\alpha = 0.05$. The p-value result is very close to zero (p-value < 0.001), indicating that BRKGA-IG performs significantly better than CSA.

According to figures 76 and 77, the proposed BRKGA-IG method outperforms the MILP, GIA, CSA, IABC, BRKGA, and IGA methods. The CSA, IABC, IGA and BRKGA algorithms have similar performances in small and medium instances

Analyzing the average of the results in the table 42 and table 43, it is possible to verify that the MILP has the highest computational time with average ARPD results, the GIA algorithm gets the worst ARPD, but with the lowest computational cost. BRKGA-IG presents the best ARPD results compared to all tested methods, with higher computational times than IGA and BRKGA and lower computational times than CSA and IABC. Finally, BRKGA-IG is considered a competitive metaheuristic so far for OSSP-VRP, with a good trade-off between solution quality and computational costs.

9.7 Final remarks

This paper aimed to develop a biased random key genetic algorithm for the open shop scheduling problem, with routing by capacitated single-vehicle. The problem's objective function is the minimization of the total time to complete the scheduling and distribution of jobs to customers (makespan).

We develop a new integer linear programming model to define the problem and solve it using a commercial solver. MILP was not able to obtain optimal solutions for $m \times n \geq 6$ size problems. It was also proposed approximate solution methods: GIA, IGA, BRKGA, and a hybrid metaheuristic (BRKGA-IG), combining the three previous methods.

The BRKGA-IG obtained better results than the constructive heuristics GIA, the classic BRKGA, the benchmarking metaheuristics CSA and IABC proposed by Feng, Niu and Liu (2021) and Li *et al.* (2020b), respectively, and the metaheuristic IGA, initially proposed by Tavares-Neto and Nagano (2019) to solve integrated problems of scheduling and distribution. BRKGA-IG, in comparison with the other approximate and exact tested methods, obtained the best ARPD in all sizes of instances, with admissible computational times.

BRKGA-IG has an intensive exploitation mechanism with iterated greedy local search procedure, a restart mechanism to reduce premature convergence of the population, and a new decoding scheme proposed for OSSP-VRP solutions; these mechanisms are one of the main contributions of the research. With these mechanisms, BRKGA-IG resulted in an average relative percentage deviation smaller than 9% and outperformed all other algorithms tested.

As an extension of this work, we also suggest studying the problem with scheduling considering explicit setup times, which is one of the main parameters present in complex industrial systems. Concerning the distribution, the routing can use multiple delivery vehicles and time window restrictions, present in several VRP problems.

Regarding the improvement of the solutions found for OSSP-VRP, it is recommended to improve the MILP, using decomposition methods, aiming to improve the efficiency and linear relaxation of the model. It is also recommended to use path-relinking together with BRKGA to improve metaheuristics since this hybridization has been obtaining good results for production scheduling problems Andrade *et al.* (2019). Finally, it is possible to verify that it can easily apply the methods developed in the article to solve other integrated production and distribution problems, including in practical instances of real industries.

10 CONCLUSIONS AND FUTURE RESEARCH

This section will illustrate the main contributions of each chapter of the doctoral dissertation and its relation to the objectives of the work presented in the introduction section. Finally, proposals for future studies will be described.

10.1 Conclusions

In this doctoral dissertation, we addressed the open shop scheduling problem. The doctoral research was the first time this problem was studied in the literature with new variants based in real industrial environments, such as operations repetition, zero buffer or machine blocking constraints, and job delivery by vehicle routing. The present study developed efficient solution methods for proposed variants considering the constraints of the cited real industrial environments. Furthermore, it was possible to develop efficient solutions methods to the problems already mentioned in the literature, such as the classical open shop problem and the variant with setup times. Finally, the proposed methods were compared with literature benchmarking methods and got competitive results.

In summary, the goal of this doctoral dissertation was to provide further insight into these important open shop problems, deeply analyzing and developing new efficient methods to solve them. In order to deal with this goal, several general research objectives were identified in Section 1.2, which have been addressed along the five parts of this doctoral dissertation as follows:

Objective 1. Implement new constructive heuristics to solve the classical open shop problem, comparing it with seminal heuristics such as priority rules and constructive heuristics presented in the literature.

In chapter 5, several efficient constructive heuristics that exploit some specific properties of the OSSP, such as operations idleness and lower bound, were proposed. An extensive computational experience using problem instances taken from the related literature was conducted to assess the performance of the proposed algorithms compared to existing ones with respect to the quality of the solutions and the CPU time required.

In summary, a new beam search and cheapest insertion procedures hybridized with new constructive heuristics were developed. Finally, an efficient local search algorithm (LS) that leads to excellent results within an admissible computational effort was proposed. The extensive computational tests show the excellent performance of the heuristics proposed, resulting in the one of best heuristics for the classic problem.

Objective 2. Use matheuristic techniques with the proposed constructive heuristics, mixed-integer linear programming, and constraint programming to

solve the variant of the problem with setup times, with randomly generated instances from the literature. Comparing the algorithms with the main state-of-the-art solving methods.

In chapter 6, a new hybridization of an ALNS with a CP model as a local search phase was presented to solve the open shop scheduling with non-anticipatory sequence-dependent setup time. ALNS-CP used an heuristic based in operations idleness proposed in the classic OSSP study as initial solution. An MILP model is also presented based on the classic open shop model, and a new CP model was proposed considering proprieties of the non-anticipatory setup times.

The proposed CP model has not been addressed in the revised literature and outperforms all other exact methods. Many approximations and exact algorithms to obtain high-quality solutions in acceptable computational times were tested. The extensive computational experience shows that the proposed hybridization of metaheuristic and constraint programming as a matheuristic approach with improvement heuristics type is promising for solving large-sized instances.

Objective 3. Propose a formal definition for the open shop problem with machine blocking and develop mathematical models of integer and constrained programming and hybrid exact methods to solve the problem.

In chapter 7, a variant of the open shop scheduling problem is considered in which the intermediate storage is forbidden among two adjacent production stages (zero buffer or machine blocking constraint). Since this is an NP-hard problem, a two-stage constraint programming approach was proposed as a new exact method.

Computational results point to the ability of the proposed method to solve large-sized instances in comparison with the developed MILP model and a simple CP model, both with user cuts. In all set of instances, the proposed two-stage method performed better than benchmarking methods and a two-stage integer programming model implemented for comparisons purposes.

Objective 4. Study the open shop scheduling problem, considering the possibility of repetition of operations, demonstrating properties of the problem, and developing exact and approximate solving methods.

In chapter 8, a new variant for the open shop scheduling problem, the open shop scheduling problem with repetitions (OSSPR), where the jobs can be processed on any machine more than once (operation by operation) was presented. Thereby, all the jobs can be scheduled in an unconstrained way, substantially increasing the number of feasible solutions in comparison with the classical open shop. The OSSPR has many applications in automotive and maintenance actives.

To solve the problem, a mixed-integer linear programming model was presented and a new constraint programming model was proposed. A new efficient variable neighborhood search is also proposed using variable search strategies through a proposed constraint programming model. Computational results show very good performance of the proposed metaheuristic on the instances tested.

Objective 5. Study the open shop problem, integrating the scheduling and distribution of orders by capacitated single vehicle and using exact methods and bio-inspired metaheuristics for the problem's solution.

In chapter 9, the open shop problem with routing by a capacitated single vehicle was proposed in attention to scheduling problems integrating production environments and distribution systems to adopt more realistic assumptions. The study presented a MILP model and a new BRKGA with an IGA local search procedure to solve the problem.

BRKGA-IG has a new decoding scheme for OSSP-VRP solutions, an intensive exploitation mechanism with an iterated greedy local search procedure, and a restart mechanism to reduce premature population convergence. With these new mechanisms, the extensive computational experience carried out showed that the proposed metaheuristic BRKGA-IG was promising in solving large-sized instances for the new proposed problem, outperforming all other tested methods.

In order to highlight the contributions already made by the doctoral dissertation, the following studies were published and have a fundamental relationship with the present research. The studies are ordered according to the doctoral dissertation's objectives sequence.

- New efficient heuristics for scheduling open shops with makespan minimization. **Computers & Operations Research**. 2022.
- A new hybridization of adaptive large neighborhood search with constraint programming for open shop scheduling with sequence-dependent setup times. **Computers & Industrial Engineering**. 2022.
- A new two-stage constraint programming approach for open shop scheduling problem with machine blocking. **International Journal of Production Research**. 2023.
- A new variable neighborhood search with constraint programming search strategy for open shop scheduling problem with repetitions of operation. **Engineering Optimization**. 2021.
- A new efficient biased random key genetic algorithm for open shop scheduling with routing by capacitated single vehicle and makespan minimization. **Engineering Applications of Artificial Intelligence**. 2021.

Therefore, this research contributed to the production scheduling field by improving the state of the art of the open shop problem, implementing new exact and approximate methods, and providing opportunities for future studies.

10.2 Future research

Some studies could improve the current literature, addressing not only the open shop mentioned variants, but other emerging variants, such as the distribution open shop, with several factory options for the jobs production (MENG *et al.*, 2020), the adoption of constraints on the jobs and machines flows, such as the characteristics of no-wait and no-idle production environments (CROCE; GROSSO; SALASSA, 2021), and the hybridization of the distribution open shop with routing problems (MOONS *et al.*, 2017).

Recently, green scheduling has received a lot of attention from operational research practitioners (BAMPIS; LETSIOS; LUCARELLI, 2015; GAHM *et al.*, 2016), and there are several opportunities in this field. Also, it is interesting to study the effect of controllable processing times (SHABTAY; STEINER, 2007; FERNANDEZ-VIAGAS; FRAMINAN, 2015a) in the constraint programming techniques on open shop variants.

Another research trend is the hybridization of constraint programming and metaheuristics in the context of decomposition matheuristics to improve efficiency to find valid solution in admissible computational times (MANIEZZO; BOSCHETTI; STÜTZLE, 2021).

Finally, the proposed metaheuristics, exact methods, and matheuristics can be extended to other manufacturing layouts, i.e., hybrid flow shop, job shop, distributed flow shop, among others. In addition, the doctoral dissertation presented extensions of the classical open shop variant, which still needed to be explored due to their practical applications in industrial and service areas. Therefore, when we compare the new proposed variants with the classical open shop problem presented in Figure 2, there was a significant growth in the production scheduling literature.

REFERENCES

ABDELMAGUID, T. F. Bi-objective dynamic multiprocessor open shop scheduling for maintenance and healthcare diagnostics. **Expert Systems with Applications**, Elsevier, v. 186, p. 115777, 2021.

ABID, T.; AYADI, O.; MASMOUDI, F. An integrated production-distribution planning problem under demand and production capacity uncertainties: New formulation and case study. **Mathematical Problems in Engineering**, Hindawi, v. 2020, 2020.

ABREU, L. R. *et al.* A genetic algorithm for scheduling open shops with sequence-dependent setup times. **Computers & Operations Research**, Elsevier, v. 113, p. 104793, 2020.

ABREU, L. R.; NAGANO, M. S. A new hybridization of adaptive large neighborhood search with constraint programming for open shop scheduling with sequence-dependent setup times. **Computers & Industrial Engineering**, Elsevier, v. 168, p. 108128, 2022.

ABREU, L. R. *et al.* New efficient heuristics for scheduling open shops with makespan minimization. **Computers & Operations Research**, Elsevier, v. 142, p. 105744, 2022.

ABREU, L. R.; TAVARES-NETO, R. F.; NAGANO, M. S. A new efficient biased random key genetic algorithm for open shop scheduling with routing by capacitated single vehicle and makespan minimization. **Engineering Applications of Artificial Intelligence**, Elsevier, v. 104, p. 104373, 2021.

ABREU, L. R. de *et al.* A new variable neighbourhood search with a constraint programming search strategy for the open shop scheduling problem with operation repetitions. **Engineering Optimization**, Taylor & Francis, p. 1–20, 2021.

ABREU, L. R. de; PRATA, B. de A. A genetic algorithm with neighborhood search procedures for unrelated parallel machine scheduling problem with sequence-dependent setup times. **Journal of Modelling in Management**, Emerald Publishing Limited, 2020.

ADAK, Z.; AKAN, M. Ö. A.; BULKAN, S. Multiprocessor open shop problem: literature review and future directions. **Journal of Combinatorial Optimization**, Springer, v. 40, p. 547–569, 2020.

AGHIGHI, S. *et al.* Open-shop production scheduling with reverse flows. **Computers & Industrial Engineering**, Elsevier, v. 153, p. 107077, 2021.

AHMADIAN, M. M. *et al.* Four decades of research on the open-shop scheduling problem to minimize the makespan. **European Journal of Operational Research**, Elsevier, 2021.

AHMADIAN, M. M. *et al.* Four decades of research on the open-shop scheduling problem to minimize the makespan. **European Journal of Operational Research**, 2021. ISSN 0377-2217. Available at: https://www.sciencedirect.com/science/article/pii/S0377221721002526.

AHMADIZAR, F.; FARAHANI, M. H. A novel hybrid genetic algorithm for the open shop scheduling problem. **The International Journal of Advanced Manufacturing Technology**, v. 62, n. 5, p. 775–787, set. 2012. ISSN 1433-3015.

AL-SALEM, A. *et al.* Scheduling to minimize makespan on unrelated parallel machines with sequence dependent setup times. **Engineering Journal of the University of Qatar**, v. 17, n. 1, p. 177–187, 2004.

ALAMEEN, M. *et al.* Improved genetic and simulating annealing algorithms to solve the traveling salesman problem using constraint programming. **Engineering Technology & Applied Science Research**, v. 6, n. 2, p. 927–930, 2016.

ALI, O.; CÔTÉ, J.-F.; COELHO, L. C. Models and algorithms for the delivery and installation routing problem. **European Journal of Operational Research**, Elsevier, 2020.

ALLAHVERDI, A. The third comprehensive survey on scheduling problems with setup times/costs. **European Journal of Operational Research**, v. 246, n. 2, p. 345–378, 2015.

ALLAHVERDI, A.; GUPTA, J. N.; ALDOWAISAN, T. A review of scheduling research involving setup considerations. **Omega**, Elsevier, v. 27, n. 2, p. 219–239, 1999.

ALLAHVERDI, A. *et al.* A survey of scheduling problems with setup times or costs. **European journal of operational research**, Elsevier, v. 187, n. 3, p. 985–1032, 2008.

ANAND, E.; PANNEERSELVAM, R. Literature review of open shop scheduling problems. **Intelligent Information Management**, v. 7, n. 1, p. 32–52, 2016.

ANAND, E.; PANNEERSELVAM, R. Development of efficient genetic algorithm for open shop scheduling problem to minimise makespan. **International Journal of Advanced Operations Management**, Inderscience Publishers (IEL), v. 10, n. 3, p. 199–233, 2018.

ANDRADE, C. E.; SILVA, T.; PESSOA, L. S. Minimizing flowtime in a flowshop scheduling problem with a biased random-key genetic algorithm. **Expert Systems with Applications**, Elsevier, v. 128, p. 67–80, 2019.

ANDRADE, C. E. *et al.* The multi-parent biased random-key genetic algorithm with implicit path-relinking and its real-world applications. **European Journal of Operational Research**, Elsevier, 2019.

ANDRESEN, M. *et al.* Simulated annealing and genetic algorithms for minimizing mean flow time in an open shop. **Mathematical and Computer Modelling**, Elsevier, v. 48, n. 7-8, p. 1279–1293, 2008.

APT, K. **Principles of constraint programming**. [*S.l.: s.n.*]: Cambridge university press, 2003.

ARAÚJO, K. A. G. de; BONATES, T. O.; PRATA, B. d. A. Modeling and scheduling hybrid open shops for makespan minimization. **Journal of Modelling in Management**, Emerald Publishing Limited, 2021.

ARCHETTI, C.; SPERANZA, M. G. A survey on matheuristics for routing problems. **EURO Journal on Computational Optimization**, Elsevier, v. 2, n. 4, p. 223–246, 2014.

ARENALES, M. *et al.* **Pesquisa operacional: para cursos de engenharia**. [*S.l.: s.n.*]: Elsevier Brasil, 2015.

AZADEH, A. *et al.* Scheduling prioritized patients in emergency department laboratories. **Computer methods and programs in biomedicine**, Elsevier, v. 117, n. 2, p. 61–70, 2014.

BABOU, N.; REBAINE, D.; BOUDHAR, M. Two-machine open shop problem with a single server and set-up time considerations. **Theoretical Computer Science**, Elsevier, v. 867, p. 13–29, 2021.

BAI, D.; TANG, L. Performance analysis of rotation schedule and improved strategy for open shop problem to minimise makespan. **International Journal of Systems Science**, Taylor & Francis, v. 42, n. 7, p. 1143–1153, 2011.

BAI, D.; TANG, L. Open shop scheduling problem to minimize makespan with release dates. **Applied Mathematical Modeling**, v. 37, n. 4, p. 2008–2015, 2013.

BAI, D.; ZHANG, Z.; ZHANG, Q. Flexible open shop scheduling problem to minimize makespan. **Computers & Operational Research**, v. 37, n. 1, p. 207–215, 2016.

BAI, D. *et al.* Open shop scheduling problem to minimize total weighted completion time. **Engineering Optimization**, Taylor & Francis, v. 49, n. 1, p. 98–112, 2017.

BAMPIS, E.; LETSIOS, D.; LUCARELLI, G. Green scheduling, flows and matchings. **Theoretical Computer Science**, Elsevier, v. 579, p. 126–136, 2015.

BEHNAMIAN, J.; DEZFOOLI, S. M.; ASGARI, H. A scatter search algorithm with a novel solution representation for flexible open shop scheduling: a multi-objective optimization. **The Journal of Supercomputing**, Springer, p. 1–24, 2021.

BEHNAMIAN, J.; GHOMI, S. F. A survey of multi-factory scheduling. **Journal of Intelligent Manufacturing**, Springer, v. 27, n. 1, p. 231–249, 2016.

BEHNEL, S. *et al.* Cython: The best of both worlds. **Computing in Science & Engineering**, Ieee, v. 13, n. 2, p. 31–39, 2011.

BEIRÃO, N. d. C. L. *et al.* **Sistema de apoio à decisão para sequenciamento de operações em ambientes Job Shop**. 1997.

BERTRAND, J. W. M.; FRANSOO, J. C. Operations management research methodologies using quantitative modeling. **International Journal of Operations & Production Management**, MCB UP Ltd, 2002.

BEVERN, R. V.; PYATKIN, A. V. Completing partial schedules for open shop with unit processing times and routing. *In*: SPRINGER. **International Computer Science Symposium in Russia**. [*S.l.: s.n.*], 2016. p. 73–87.

BIRGIN, E. G.; FERREIRA, J. E.; RONCONI, D. P. A filtered beam search method for the m-machine permutation flowshop scheduling problem minimizing the earliness and tardiness penalties and the waiting time of the jobs. **Computers & Operations Research**, Elsevier, v. 114, p. 104824, 2020.

BLAZEWICZ, J.; KOVALYOV, M. Y. The complexity of two group scheduling problems. **Journal of Scheduling**, Wiley Online Library, v. 5, n. 6, p. 477–485, 2002.

BLUM, C. Beam-aco–hybridizing ant colony optimization with beam search: an application to open shop scheduling. **Computers & Operations Research**, v. 32, n. 6, p. 1565–1591, 2005.

BLUM, C.; ROLI, A. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. **ACM computing surveys (CSUR)**, Acm, v. 35, n. 3, p. 268–308, 2003.

BOUZIDI, A.; RIFFI, M. E.; BARKATOU, M. Cat swarm optimization for solving the open shop scheduling problem. **Journal of Industrial Engineering International**, Springer, v. 15, n. 2, p. 367–378, 2019.

BRÄSEL, H.; TAUTENHAHN, T.; WERNER, F. Constructive heuristic algorithms for the open shop problem. **Computing**, v. 51, n. 2, p. 95–110, jun. 1993. ISSN 1436-5057.

BRUCKER, P. *et al.* A branch and bound algorithm for the open-shop problem. **Discrete Applied Mathematics**, v. 76, n. 1, p. 43–59, 1997.

BRUCKER, P. *et al.* Complexity results for flow-shop and open-shop scheduling problems with transportation delays. **Annals of Operations Research**, Springer, v. 129, n. 1-4, p. 81–106, 2004.

BUER, M. G. V.; WOODRUFF, D. L.; OLSON, R. T. Solving the medium newspaper production/distribution problem. **European Journal of Operational Research**, Elsevier, v. 115, n. 2, p. 237–253, 1999.

CANKAYA, B.; WARI, E.; TOKGOZ, B. E. Practical approaches to chemical tanker scheduling in ports: a case study on the port of houston. **Maritime Economics & Logistics**, Springer, v. 21, n. 4, p. 559–575, 2019.

CHALESHTARTI, A. S. *et al.* A hybrid genetic and lagrangian relaxation algorithm for resource-constrained project scheduling under nonrenewable resources. **Applied Soft Computing**, Elsevier, v. 94, p. 106482, 2020.

CHANG, Y.-C.; LI, V. C.; CHIANG, C.-J. An ant colony optimization heuristic for an integrated production and distribution scheduling problem. **Engineering Optimization**, Taylor & Francis, v. 46, n. 4, p. 503–520, 2014.

CHAVES, A. A.; GONÇALVES, J. F.; LORENA, L. A. N. Adaptive biased random-key genetic algorithm with local search for the capacitated centered clustering problem. **Computers & Industrial Engineering**, Elsevier, v. 124, p. 331–346, 2018.

CHEN, C.; DEMIR, E.; HUANG, Y. An adaptive large neighborhood search heuristic for the vehicle routing problem with time windows and delivery robots. **European Journal of Operational Research**, Elsevier, 2021.

CHEN, X. *et al.* Customized bus route design with pickup and delivery and time windows: Model, case study and comparative analysis. **Expert Systems with Applications**, Elsevier, p. 114242, 2020.

CHENG, B.-Y.; LEUNG, J. Y.; LI, K. Integrated scheduling on a batch machine to minimize production, inventory and distribution costs. **European Journal of Operational Research**, Elsevier, v. 258, n. 1, p. 104–112, 2017.

CHERRI, L. H. Programação por restrições: Um breve tutorial. **Pesquisa Operacional para o Desenvolvimento**, v. 10, n. 1, p. 1–8, 2018.

COBAN, R. A context layered locally recurrent neural network for dynamic system identification. **Engineering Applications of Artificial Intelligence**, Elsevier, v. 26, n. 1, p. 241–250, 2013.

COBAN, R.; AKSU, I. O. Neuro-controller design by using the multifeedback layer neural network and the particle swarm optimization. **Tehnički vjesnik**, Strojarski fakultet u Slavonskom Brodu; Fakultet elektrotehnike, računarstva . . . , v. 25, n. 2, p. 437–444, 2018.

COBAN, R.; CAN, B. A trajectory tracking genetic fuzzy logic controller for nuclear research reactors. **Energy Conversion and Management**, Elsevier, v. 51, n. 3, p. 587–593, 2010.

COLAK, S.; AGARWAL, A. Non-greedy heuristics and augmented neural networks for the open-shop scheduling problem. **Naval Research Logistics (NRL)**, Wiley Online Library, v. 52, n. 7, p. 631–644, 2005.

CONWAY, R.; MAXWELL, W.; MILLER, L. **Theory of Scheduling**. Reading: Addison-Wesley, 2003.

COTA, L. P. *et al.* An adaptive multi-objective algorithm based on decomposition and large neighborhood search for a green machine scheduling problem. **Swarm and Evolutionary Computation**, Elsevier, v. 51, p. 100601, 2019.

CROCE, F. D.; GROSSO, A.; SALASSA, F. Minimizing total completion time in the two-machine no-idle no-wait flow shop problem. **Journal of Heuristics**, Springer, p. 1–15, 2021.

CROCE, F. D.; NARAYAN, V.; TADEI, R. The two-machine total completion time flow shop problem. **European Journal of Operational Research**, Elsevier, v. 90, n. 2, p. 227–237, 1996.

DABAH, A. *et al.* Efficient parallel tabu search for the blocking job shop scheduling problem. **Soft Computing**, Springer, v. 23, n. 24, p. 13283–13295, 2019.

DARVISH, M.; COELHO, L. C. Sequential versus integrated optimization: Production, location, inventory control, and distribution. **European Journal of Operational Research**, Elsevier, v. 268, n. 1, p. 203–214, 2018.

DESROCHERS, M.; LAPORTE, G. Improvements and extensions to the miller-tucker-zemlin subtour elimination constraints. **Operations Research Letters**, Elsevier, v. 10, n. 1, p. 27–36, 1991.

DEVAPRIYA, P.; FERRELL, W.; GEISMAR, N. Optimal fleet size of an integrated production and distribution scheduling problem for a perishable product. *In*: INSTITUTE OF INDUSTRIAL AND SYSTEMS ENGINEERS (IISE). **IIE Annual Conference. Proceedings**. [*S.l.: s.n.*], 2006. p. 1.

DONG, X.; HUANG, H.; CHEN, P. An improved neh-based heuristic for the permutation flowshop problem. **Computers & Operations Research**, Elsevier, v. 35, n. 12, p. 3962–3968, 2008.

FAN, K. *et al.* Review and classification of hybrid shop scheduling. **Production Engineering**, v. 12, n. 5, p. 597–609, out. 2018. ISSN 1863-7353.

FARAHANI, P.; GRUNOW, M.; GÜNTHER, H.-O. Integrated production and distribution planning for perishable food products. **Flexible services and manufacturing journal**, Springer, v. 24, n. 1, p. 28–51, 2012.

FENG, Z.-k.; NIU, W.-j.; LIU, S. Cooperation search algorithm: A novel metaheuristic evolutionary intelligence algorithm for numerical optimization and engineering optimization problems. **Applied Soft Computing**, Elsevier, v. 98, p. 106734, 2021.

FERNANDES, F. C. F.; FILHO, M. G. **Planejamento e controle da produção: dos fundamentos ao essencial**. [*S.l.: s.n.*]: Atlas São Paulo, 2010.

FERNANDEZ-VIAGAS, V.; FRAMINAN, J. M. Controllable processing times in project and production management: Analysing the trade-off between processing times and the amount of resources. **Mathematical Problems in Engineering**, Hindawi, v. 2015, 2015.

FERNANDEZ-VIAGAS, V.; FRAMINAN, J. M. Efficient non-population-based algorithms for the permutation flowshop scheduling problem with makespan minimisation subject to a maximum tardiness. **Computers & Operations Research**, v. 64, p. 86–96, 2015.

FERNANDEZ-VIAGAS, V.; RUIZ, R.; FRAMINAN, J. M. A new vision of approximate methods for the permutation flowshop to minimise makespan: State-of-the-art and computational evaluation. **European Journal of Operational Research**, v. 257, n. 3, p. 707–721, 2017. ISSN 0377-2217. Available at: http://www.sciencedirect.com/science/article/pii/S0377221716308074.

FISCHETTI, M.; FISCHETTI, M. Matheuristics. *In*: **Handbook of heuristics**. [*S.l.: s.n.*]: Springer, 2018. p. 121–153.

FRAMINAN, J. M.; LEISTEN, R.; GARCÍA, R. R. **Manufacturing scheduling systems**. [*S.l.: s.n.*]: Springer, 2014. 51–63 p.

FRAMINAN, J. M.; PEREZ-GONZALEZ, P. New approximate algorithms for the customer order scheduling problem with total completion time objective. **Computers & Operations Research**, v. 78, p. 181–192, 2017.

FRAMINAN, J. M.; PEREZ-GONZALEZ, P. Order scheduling with tardiness objective: Improved approximate solutions. **European Journal of Operational Research**, v. 266, n. 3, p. 840–850, 2018. ISSN 0377-2217. Available at: https://www.sciencedirect.com/science/article/pii/S0377221717309931.

FRIEDRICH, C.; ELBERT, R. Adaptive large neighborhood search for vehicle routing problems with transshipment facilities arising in city logistics. **Computers & Operations Research**, Elsevier, v. 137, p. 105491, 2022.

FU, Y. *et al.* Bi-objective modeling and optimization for stochastic two-stage open shop scheduling problems in the sharing economy. **IEEE Transactions on Engineering Management**, IEEE, 2021.

FUCHIGAMI, H. Y.; MOCCELLIN, J. V.; RUIZ, R. Novas regras de prioridade para programação em flexible flow line com tempos de setup explícitos. **Production**, SciELO Brasil, v. 25, n. 4, p. 779–790, 2015.

GAHM, C. *et al.* Energy-efficient scheduling in manufacturing companies: A review and research framework. **European Journal of Operational Research**, Elsevier, v. 248, n. 3, p. 744–757, 2016.

GANJI, M. *et al.* A green multi-objective integrated scheduling of production and distribution with heterogeneous fleet vehicle routing and time windows. **Journal of Cleaner Production**, Elsevier, p. 120824, 2020.

GAO, S.; QI, L.; LEI, L. Integrated batch production and distribution scheduling with limited vehicle capacity. **International Journal of Production Economics**, Elsevier, v. 160, p. 13–25, 2015.

GAREY, M.; JOHNSON, D. **Computers and Intractability: a guide to the theory of NP-completeness**. Norwell: Freeman, 2012.

GASPAR-CUNHA, A.; TAKAHASHI, R.; ANTUNES, C. H. **Manual de computação evolutiva e metaheurística**. [*S.l.: s.n.*]: Imprensa da Universidade de Coimbra/Coimbra University Press, 2012.

GEDIK, R. *et al.* A constraint programming approach for solving unrelated parallel machine scheduling problem. **Computers & Industrial Engineering**, Elsevier, v. 121, p. 139–149, 2018.

GHANNADPOUR, S. F.; ZANDIYEH, F. An adapted multi-objective genetic algorithm for solving the cash in transit vehicle routing problem with vulnerability estimation for risk quantification. **Engineering Applications of Artificial Intelligence**, Elsevier, v. 96, p. 103964, 2020.

GHOSN, S. B.; DROUBY, F.; HARMANANI, H. M. A parallel genetic algorithm for the open-shop scheduling problem using deterministic and random moves. **Int. J. Artif. Intell**, v. 14, n. 1, p. 130–144, 2016.

GOLDANSAZ, S. M.; JOLAI, F.; ANARAKI, A. H. Z. A hybrid imperialist competitive algorithm for minimizing makespan in a multi-processor open shop. **Applied Mathematical Modelling**, Elsevier, v. 37, n. 23, p. 9603–9616, 2013.

GONÇALVES, J. F.; MENDES, J. J. de M.; RESENDE, M. G. A hybrid genetic algorithm for the job shop scheduling problem. **European journal of operational research**, Elsevier, v. 167, n. 1, p. 77–95, 2005.

GONÇALVES, J. F.; RESENDE, M. G. An evolutionary algorithm for manufacturing cell formation. **Computers & industrial engineering**, Elsevier, v. 47, n. 2-3, p. 247–273, 2004.

GONÇALVES, J. F.; RESENDE, M. G. Biased random-key genetic algorithms for combinatorial optimization. **Journal of Heuristics**, Springer, v. 17, n. 5, p. 487–525, 2011.

GONG, D.; HAN, Y.; SUN, J. A novel hybrid multi-objective artificial bee colony algorithm for blocking lot-streaming flow shop scheduling problems. **Knowledge-Based Systems**, Elsevier, v. 148, p. 115–130, 2018.

GONZÁLEZ-RODRÍGUEZ, I. *et al.* Heuristic local search for fuzzy open shop scheduling. *In*: IEEE. **International Conference on Fuzzy Systems**. [*S.l.: s.n.*], 2010. p. 1–8.

GONZALEZ, T.; SAHNI, S. Open shop scheduling to minimize finish time. **Journal of the Association for Computing Machinery**, Acm, v. 23, n. 4, p. 665–679, 1976.

GRAHAM, R. L. *et al.* Optimization and approximation in deterministic sequencing and scheduling: a survey. *In*: **Annals of discrete mathematics**. [*S.l.: s.n.*]: Elsevier, 1979. v. 5, p. 287–326.

GUÉRET, C.; JUSSIEN, N.; PRINS, C. Using intelligent backtracking to improve branch-and-bound methods: An application to open-shop problems. **European Journal of Operational Research**, v. 127, n. 2, p. 165–183, 2000.

GUÉRET, C.; PRINS, C. Classical and new heuristics for the open-shop problem. **European Journal of Operational Research**, v. 107, n. 2, p. 306–314, 1998.

GUÉRET, C.; PRINS, C. A new lower bound for the open shop problem. **Annals of Operations Research**, v. 92, n. 0, p. 165–183, 1999.

HÀ, M. H. *et al.* A new constraint programming model and a linear programming-based adaptive large neighborhood search for the vehicle routing problem with synchronization constraints. **Computers & Operations Research**, Elsevier, v. 124, p. 105085, 2020.

HALL, N. G.; SRISKANDARAJAH, C. A survey of machine scheduling problems with blocking and no-wait in process. **Operations research**, Informs, v. 44, n. 3, p. 510–525, 1996.

HAN, Y. *et al.* Evolutionary multi-objective blocking lot-streaming flow shop scheduling with interval processing time. **Applied Soft Computing**, Elsevier, v. 42, p. 229–245, 2016.

HAN, Y. *et al.* Evolutionary multiobjective blocking lot-streaming flow shop scheduling with machine breakdowns. **IEEE transactions on cybernetics**, Ieee, v. 49, n. 1, p. 184–197, 2017.

HANSEN, P.; MLADENOVIĆ, N.; PÉREZ, J. A. M. Variable neighbourhood search: methods and applications. **Annals of Operations Research**, Springer, v. 175, n. 1, p. 367–407, 2010.

HANSEN, P. *et al.* Variable neighborhood search: basics and variants. **EURO Journal on Computational Optimization**, Springer, v. 5, n. 3, p. 423–454, 2017.

HARDIN, J. R.; NEMHAUSER, G. L.; SAVELSBERGH, M. W. Strong valid inequalities for the resource-constrained scheduling problem with uniform resource requirements. **Discrete Optimization**, Elsevier, v. 5, n. 1, p. 19–35, 2008.

HARMANANI, H. M.; GHOSN, S. B. An efficient method for the open-shop scheduling problem using simulated annealing. *In*: **Information technology: New generations**. [*S.l.: s.n.*]: Springer, 2016. p. 1183–1193.

HE, L.; WEERDT, M. de; YORKE-SMITH, N. Time/sequence-dependent scheduling: the design and evaluation of a general purpose tabu-based adaptive large neighbourhood search algorithm. **Journal of Intelligent Manufacturing**, Springer, p. 1–28, 2019.

HILLIER, F. S.; LIEBERMAN, G. J. **Introdução à pesquisa operacional**. [*S.l.: s.n.*]: McGraw Hill Brasil, 2013.

HOJABRI, H. *et al.* Large neighborhood search with constraint programming for a vehicle routing problem with synchronization constraints. **Computers & Operations Research**, Elsevier, v. 92, p. 87–97, 2018.

HONG, I.-H.; CHOU, C.-C.; LEE, P.-K. Admission control in queue-time loop production-mixed integer programming with lagrangian relaxation (miplar). **Computers & Industrial Engineering**, Elsevier, v. 129, p. 417–425, 2019.

HOSSEINABADI, A. A. R. *et al.* Extended genetic algorithm for solving open-shop scheduling problem. **Soft Computing**, abr. 2018. ISSN 1433-7479.

JIN, F.; SONG, S.; WU, C. An improved version of the neh algorithm and its application to large-scale flow-shop scheduling problems. **Iie Transactions**, Taylor & Francis, v. 39, n. 2, p. 229–234, 2007.

KANG, L.; CHEN, S.; MENG, Q. Bus and driver scheduling with mealtime windows for a single public bus route. **Transportation Research Part C: Emerging Technologies**, Elsevier, v. 101, p. 145–160, 2019.

KELBEL, J.; HANZÁLEK, Z. Solving production scheduling with earliness/tardiness penalties by constraint programming. **Journal of Intelligent Manufacturing**, Springer, v. 22, n. 4, p. 553–562, 2011.

KHURI, S.; MIRYALA, S. Genetic algorithms for solving open shop scheduling problems. **Progress in Artificial Intelligence**, Springer, p. 849–849, 1999.

KIZILAY, D.; ÇIL, Z. A. Constraint programming approach for multi-objective two-sided assembly line balancing problem with multi-operator stations. **Engineering Optimization**, Taylor & Francis, p. 1–16, 2020.

KIZILAY, D. *et al.* A variable block insertion heuristic for solving permutation flow shop scheduling problem with makespan criterion. **Algorithms**, Multidisciplinary Digital Publishing Institute, v. 12, n. 5, p. 100, 2019.

KURDI, M. Ant colony optimization with a new exploratory heuristic information approach for open shop scheduling problem. **Knowledge-Based Systems**, Elsevier, v. 22, p. 001162, 2022.

LABORIE, P. An update on the comparison of mip, cp and hybrid approaches for mixed resource allocation and scheduling. *In*: SPRINGER. **International conference on the integration of constraint programming, artificial intelligence, and operations research**. [*S.l.: s.n.*], 2018. p. 403–411.

LABORIE, P.; GODARD, D. Self-adapting large neighborhood search: Application to single-mode scheduling problems. **Proceedings MISTA-07, Paris**, Citeseer, v. 8, 2007.

LABORIE, P. *et al.* Ibm ilog cp optimizer for scheduling. **Constraints**, Springer, v. 23, n. 2, p. 210–250, 2018.

LAKATOS, E.; MARCONI, M. de andrade. **Fundamentos de metodologia científica: Técnicas de pesquisa**, v. 7, 2010.

LANGE, J.; WERNER, F. On neighborhood structures and repair techniques for blocking job shop scheduling problems. **Algorithms**, Multidisciplinary Digital Publishing Institute, v. 12, n. 11, p. 242, 2019.

LATORRE, A. *et al.* Fairness in bio-inspired optimization research: A prescription of methodological guidelines for comparing meta-heuristics. **arXiv preprint arXiv:2004.09969**, 2020.

LAWLER, E. L. *et al.* Sequencing and scheduling: Algorithms and complexity. **Handbooks in operations research and management science**, Elsevier, v. 4, p. 445–522, 1993.

LEE, Y. G.; MALONE, M. F. Flexible batch process planning. **Industrial & Engineering Chemistry Research**, ACS Publications, v. 39, n. 6, p. 2045–2055, 2000.

LEVNER, E. Optimal planning of parts' machining on a number of machines. **Automatin and Remote Control**, v. 12, n. 12, p. 1972–1978, 1969.

LI, J. *et al.* Improved artificial immune system algorithm for type-2 fuzzy flexible job shop scheduling problem. **IEEE Transactions on Fuzzy Systems**, Ieee, 2020.

LI, J.-q. *et al.* Meta-heuristic algorithm for solving vehicle routing problems with time windows and synchronized visit constraints in prefabricated systems. **Journal of Cleaner Production**, Elsevier, v. 250, p. 119464, 2020.

LI, S.; NEGENBORN, R. R.; LODEWIJKS, G. Planning inland vessel operations in large seaports using a two-phase approach. **Computers & Industrial Engineering**, Elsevier, v. 106, p. 41–57, 2017.

LI, Y.; CHEN, M.; HUO, J. A hybrid adaptive large neighborhood search algorithm for the large-scale heterogeneous container loading problem. **Expert Systems with Applications**, Elsevier, v. 189, p. 115909, 2022.

LI, Y.-Z. *et al.* An adaptive iterated greedy algorithm for distributed mixed no-idle permutation flowshop scheduling problems. **Swarm and Evolutionary Computation**, Elsevier, v. 63, p. 100874, 2021.

LIAW, C. A tabu search algorithm for the open shop scheduling problem. **Computers and Operations Research**, v. 52, n. 2, p. 109–126, 1999.

LIAW, C.-F. An iterative improvement approach for the nonpreemptive open shop scheduling problem. **European Journal of Operational Research**, v. 111, n. 3, p. 509–517, 1998.

LIAW, C.-F. A hybrid genetic algorithm for the open shop scheduling problem. **European Journal of Operational Research**, v. 124, n. 1, p. 28–42, 2000.

LIAW, C. fang. Applying simulated annealing to the open shop scheduling problem. **IIE Transactions**, v. 31, n. 5, p. 457–465, 1999.

LIN, H.-T.; LEE, H.-T.; PAN, W.-J. Heuristics for scheduling in a no-wait open shop with movable dedicated machines. **International Journal of Production Economics**, v. 111, n. 2, p. 368–377, 2008. ISSN 0925-5273. Special Section on Sustainable Supply Chain. Available at: http://www.sciencedirect.com/science/article/pii/S0925527307000515.

LIN, S.-W.; YING, K.-C. Optimization of makespan for no-wait flowshop scheduling problems using efficient matheuristics. **Omega**, Elsevier, v. 64, p. 115–125, 2016.

LÓPEZ-IBÁNEZ, M. *et al.* The irace package: Iterated racing for automatic algorithm configuration. **Operations Research Perspectives**, Elsevier, v. 3, p. 43–58, 2016.

LUNARDI, W. T. *et al.* Mixed integer linear programming and constraint programming models for the online printing shop scheduling problem. **Computers & Operations Research**, Elsevier, v. 123, p. 105020, 2020.

MACCARTHY, B. L.; LIU, J. Addressing the gap in scheduling research: a review of optimization and heuristic methods in production scheduling. **The International Journal of Production Research**, Taylor & Francis, v. 31, n. 1, p. 59–79, 1993.

MALAPERT, A. *et al.* An optimal constraint programming approach to the open-shop problem. **INFORMS Journal on Computing**, Informs, v. 24, n. 2, p. 228–244, 2012.

MANIEZZO, V.; BOSCHETTI, M.; STÜTZLE, T. **Matheuristics: Algorithms and Implementations (EURO Advanced Tutorials on Operational Research)**. [*S.l.: s.n.*]: Springer: New York, NY, USA, 2021.

MANIEZZO, V.; STÜTZLE, T.; VOSS, S. **Hybridizing Metaheuristics and Mathematical Programming. Series: Annals of Information Systems**. [*S.l.: s.n.*]: Springer, New York, 2009.

MASCIS, A.; PACCIARELLI, D. Job-shop scheduling with blocking and no-wait constraints. **European Journal of Operational Research**, Elsevier, v. 143, n. 3, p. 498–517, 2002.

MEJÍA, G.; CABALLERO-VILLALOBOS, J. P.; MONTOYA, C. Petri nets and deadlock-free scheduling of open shop manufacturing systems. **IEEE Transactions on Systems, Man, and Cybernetics: Systems**, Ieee, v. 48, n. 6, p. 1017–1028, 2018.

MEJÍA, G.; YURASZECK, F. A self-tuning variable neighborhood search algorithm and an effective decoding scheme for open shop scheduling problems with travel/setup times. **European Journal of Operational Research**, Elsevier, 2020.

MENG, L. *et al.* Mixed-integer linear programming and constraint programming formulations for solving distributed flexible job shop scheduling problem. **Computers & Industrial Engineering**, Elsevier, v. 142, p. 106347, 2020.

MIGUEL, P. A. C. *et al.* **Metodologia de pesquisa em engenharia de produção e gestão de operações**. [*S.l.: s.n.*]: Rio de Janeiro: Elsevier, 2010.

MIN, H.; ZHOU, G. Supply chain modeling: past, present and future. **Computers & industrial engineering**, Elsevier, v. 43, n. 1-2, p. 231–249, 2002.

MIYATA, H. H.; NAGANO, M. S. The blocking flow shop scheduling problem: A comprehensive and conceptual review. **Expert Systems with Applications**, v. 137, p. 130–156, 2019. ISSN 0957-4174. Available at: https://www.sciencedirect.com/science/article/pii/S0957417419304774.

MLADENOVIĆ, N.; HANSEN, P. Variable neighborhood search. **Computers & operations research**, Elsevier, v. 24, n. 11, p. 1097–1100, 1997.

MOCCELLIN, J. V. *et al.* Heuristic algorithms for scheduling hybrid flow shops with machine blocking and setup times. **Journal of the Brazilian Society of Mechanical Sciences and Engineering**, v. 40, n. 2, p. 40, jan. 2018. ISSN 1806-3691.

MOGALI, J. K.; BARBULESCU, L.; SMITH, S. F. Efficient primal heuristic updates for the blocking job shop problem. **European Journal of Operational Research**, Elsevier, 2021.

MOLINA, J. C. *et al.* The heterogeneous vehicle routing problem with time windows and a limited number of resources. **Engineering Applications of Artificial Intelligence**, Elsevier, v. 94, p. 103745, 2020.

MONTGOMERY, D. C. **Design and analysis of experiments**. [*S.l.: s.n.*]: John Wiley & Sons, 2017.

MOONS, S. *et al.* Integrating production scheduling and vehicle routing decisions at the operational decision level: a review and discussion. **Computers & Industrial Engineering**, Elsevier, v. 104, p. 224–245, 2017.

MOSHEIOV, G. *et al.* Two-machine flow shop and open shop scheduling problems with a single maintenance window. **European Journal of Operational Research**, v. 271, n. 2, p. 388–400, 2018. ISSN 0377-2217. Available at: http://www.sciencedirect.com/science/article/pii/S0377221718303229.

MOUSAVI, M.; HAJIAGHAEI-KESHTELI, M.; TAVAKKOLI-MOGHADDAM, R. Two calibrated meta-heuristics to solve an integrated scheduling problem of production and air transportation with the interval due date. **Soft Computing**, Springer, p. 1–29, 2020.

NADERI, B. *et al.* A contribution and new heuristics for open shop scheduling. **Computers & Operations Research**, v. 37, n. 1, p. 213–221, 2010. ISSN 0305-0548. Available at: http://www.sciencedirect.com/science/article/pii/S0305054809001208.

NADERI, B. *et al.* Modeling and scheduling open shops with sequence-dependent setup times to minimize total completion time. **The International Journal of Advanced Manufacturing Technology**, Springer, v. 53, n. 5-8, p. 751–760, 2011.

NADERI, B. *et al.* A study on open shop scheduling to minimise total tardiness. **International Journal of Production Research**, Taylor & Francis, v. 49, n. 15, p. 4657–4678, 2011.

NADERI, B.; NAJAFI, E.; YAZDANI, M. An electromagnetism-like metaheuristic for open-shop problems with no buffer. **Journal of Industrial Engineering International**, Springer, v. 8, n. 1, p. 29, 2012.

NADERI, B.; ZANDIEH, M. Modeling and scheduling no-wait open shop problems. **International Journal of Production Economics**, v. 158, n. 1, p. 256–266, 2014.

NAWAZ, M.; JR, E. E. E.; HAM, I. A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. **Omega**, Elsevier, v. 11, n. 1, p. 91–95, 1983.

NIEDERBERGER, C. **Chemmacros**: comprehensive support for typesetting chemistry documents. [*S.l.: s.n.*], 2012. V5.8a. Available at: http://linorg.usp.br/CTAN/macros/latex/contrib/chemmacros/chemmacros\%5Fen.pdf. Access at: 26 jun. 2017.

NISHI, T.; HIRANAKA, Y. Lagrangian relaxation and cut generation for sequence-dependent setup time flowshop scheduling problems to minimise the total weighted tardiness. **International Journal of Production Research**, Taylor & Francis, v. 51, n. 16, p. 4778–4796, 2013.

NIU, W.-j. *et al.* Multiple hydropower reservoirs operation by hyperbolic grey wolf optimizer based on elitism selection and adaptive mutation. **Water Resources Management**, Springer, v. 35, n. 2, p. 573–591, 2021.

OZER, E. A.; SARAC, T. Mip models and a matheuristic algorithm for an identical parallel machine scheduling problem under multiple copies of shared resources constraints. **Top**, Springer, v. 27, n. 1, p. 94–124, 2019.

OZOLINS, A. Dynamic programming approach for solving the open shop problem. **Central European Journal of Operations Research**, jun. 2019. ISSN 1613-9178.

ÖZTOP, H. *et al.* Metaheuristics with restart and learning mechanisms for the no-idle flowshop scheduling problem with makespan criterion. **Computers & Operations Research**, Elsevier, p. 105616, 2021.

PALACIOS, J. J. *et al.* Swarm lexicographic goal programming for fuzzy open shop scheduling. **Journal of Intelligent Manufacturing**, Springer, v. 26, n. 6, p. 1201–1215, 2015.

PANWALKAR, S.; KOULAMAS, C. The two-machine no-wait general and proportionate open shop makespan problem. **European Journal of Operational Research**, Elsevier, v. 238, n. 2, p. 471–475, 2014.

PETCH, R. J.; SALHI, S. A multi-phase constructive heuristic for the vehicle routing problem with multiple trips. **Discrete Applied Mathematics**, Elsevier, v. 133, n. 1-3, p. 69–92, 2003.

PINARBAŞI, M. New mathematical and constraint programming models for u-type assembly line balancing problems with assignment restrictions. **Engineering Optimization**, Taylor & Francis, p. 1–16, 2021.

PINEDO, M. **Scheduling: theory, algorithms, and systems**. New York: Prentice-Hall, 2016.

PISINGER, D.; ROPKE, S. A general heuristic for vehicle routing problems. **Computers & operations research**, Elsevier, v. 34, n. 8, p. 2403–2435, 2007.

PONGCHAIRERKS, P.; KACHITVICHYANUKUL, V. A two-level particle swarm optimisation algorithm for open-shop scheduling problem. **International Journal of Computing Science and Mathematics**, Inderscience Publishers (IEL), v. 7, n. 6, p. 575–585, 2016.

PRATA, B. A.; RODRIGUES, C. D.; FRAMINAN, J. M. Customer order scheduling problem to minimize makespan with sequence-dependent setup times. **Computers & Industrial Engineering**, v. 151, p. 106962, 2021. ISSN 0360-8352. Available at: https://www.sciencedirect.com/science/article/pii/S0360835220306355.

PRATA, B. de A.; ABREU, L. R. de; LIMA, J. Y. F. Heuristic methods for the single-machine scheduling problem with periodical resource constraints. **Top**, Springer, p. 1–23, 2020.

PRINS, C. Competitive genetic algorithms for the open-shop scheduling problem. **Mathematical Methods of Operations Research**, v. 52, n. 3, p. 389–411, 2000.

RAMUDHIN, A.; MARIER, P. The generalized shifting bottleneck procedure. **European Journal of Operational Research**, Elsevier, v. 93, n. 1, p. 34–48, 1996.

ROHANINEJAD, M.; HANZÁLEK, Z.; TAVAKKOLI-MOGHADDAM, R. Scheduling of parallel 3d-printing machines with incompatible job families: A matheuristic algorithm. *In*: SPRINGER. **IFIP International Conference on Advances in Production Management Systems**. [*S.l.: s.n.*], 2021. p. 51–61.

ROSHANAEI, V.; ESFEHANI, M.; ZANDIEH, M. Integrating non-preemptive open shops scheduling with sequence-dependent setup times using advanced metaheuristics. **Expert Systems with Applications**, v. 37, n. 1, p. 259–266, 2010.

ROSSI, F.; BEEK, P. V.; WALSH, T. **Handbook of constraint programming**. [*S.l.: s.n.*]: Elsevier, 2006.

ROSSI, F. L.; NAGANO, M. S. Heuristics and metaheuristics for the mixed no-idle flowshop with sequence-dependent setup times and total tardiness minimisation. **Swarm and Evolutionary Computation**, Elsevier, p. 100689, 2020.

ROSTAMI, M.; NIKRAVESH, S.; SHAHIN, M. Minimizing total weighted completion and batch delivery times with machine deterioration and learning effect: a case study from wax production. **Operational Research**, Springer, p. 1–33, 2018.

ROTHLAUF, F. Optimization methods. *In*: **Design of Modern Heuristics**. [*S.l.: s.n.*]: Springer, 2011. p. 45–102.

RUIZ, R.; STÜTZLE, T. An iterated greedy heuristic for the sequence dependent setup times flowshop problem with makespan and weighted tardiness objectives. **European Journal of Operational Research**, Elsevier, v. 187, n. 3, p. 1143–1159, 2008.

SABERI-ALIABAD, H.; REISI-NAFCHI, M.; MOSLEHI, G. Energy-efficient scheduling in an unrelated parallel-machine environment under time-of-use electricity tariffs. **Journal of Cleaner Production**, Elsevier, v. 249, p. 119393, 2020.

SACRAMENTO, D.; SOLNON, C.; PISINGER, D. Constraint programming and local search heuristic: a matheuristic approach for routing and scheduling feeder vessels in multi-terminal ports. *In*: SPRINGER. **SN Operations Research Forum**. [*S.l.: s.n.*], 2020. v. 1, p. 1–33.

SALIH, S. Q.; ALSEWARI, A. A. A new algorithm for normal and large-scale optimization problems: Nomadic people optimizer. **Neural Computing and Applications**, Springer, v. 32, n. 14, p. 10359–10386, 2020.

SHA, D.; HSU, C. A new particle swarm optimization for the open shop scheduling problem. **Computers and Operations Research**, v. 35, n. 10, p. 3243–3261, 2008.

SHA, D.; HSU, C.-Y. A hybrid particle swarm optimization for job shop scheduling problem. **Computers & Industrial Engineering**, Elsevier, v. 51, n. 4, p. 791–808, 2006.

SHABTAY, D.; STEINER, G. A survey of scheduling with controllable processing times. **Discrete Applied Mathematics**, Elsevier, v. 155, n. 13, p. 1643–1666, 2007.

SHAREH, M. B. *et al.* An improved bat optimization algorithm to solve the tasks scheduling problem in open shop. **Neural Computing and Applications**, Springer, v. 33, n. 5, p. 1559–1573, 2021.

SHAW, P. Using constraint programming and local search methods to solve vehicle routing problems. *In*: SPRINGER. **International conference on principles and practice of constraint programming**. [*S.l.: s.n.*], 1998. p. 417–431.

SHEIKHALISHAHI, M. *et al.* Multi-objective open shop scheduling by considering human error and preventive maintenance. **Applied Mathematical Modelling**, v. 67, p. 573–587, 2019. ISSN 0307-904x. Available at: http://www.sciencedirect.com/science/article/pii/S0307904X18305444.

SIDNEY, J. B.; SRISKANDARAJAH, C. A heuristic for the two-machine no-wait openshop scheduling problem. **Naval Research Logistics (NRL)**, Wiley Online Library, v. 46, n. 2, p. 129–145, 1999.

SKRIVER, A. J.; ANDERSEN, K. A. A label correcting approach for solving bicriterion shortest-path problems. **Computers & Operations Research**, Elsevier, v. 27, n. 6, p. 507–524, 2000.

SOARES, L. C. R.; CARVALHO, M. A. M. Biased random-key genetic algorithm for scheduling identical parallel machines with tooling constraints. **European Journal of Operational Research**, Elsevier, 2020.

STAFFORD, E.; TSENG, F. T.; GUPTA, J. N. Comparative evaluation of milp flowshop models. **Journal of the Operational Research Society**, Springer, v. 56, n. 1, p. 88–101, 2005.

STRUSEVICH, V. A. Two machine open shop scheduling problem with setup, processing and removal times separated. **Computers & operations research**, Elsevier, v. 20, n. 6, p. 597–611, 1993.

STRUSEVICH, V. A. A greedy open shop heuristic with job priorities. **Annals of operations research**, Springer, v. 83, p. 253–270, 1998.

STRUSEVICH, V. A. A heuristic for the two-machine open-shop scheduling problem with transportation times. **Discrete Applied Mathematics**, Elsevier, v. 93, n. 2-3, p. 287–304, 1999.

SU, L.-H.; HSIAO, M.-C. Two-agent scheduling in open shops subject to machine availability and eligibility constraints. **Journal of Industrial Engineering and Management**, v. 8, n. 4, p. 1103–1124, 2015.

TAILLARD, E. Benchmarks for basic scheduling problems. **European Journal of Operational Research**, Elsevier, v. 64, n. 2, p. 278–285, 1993.

TAVARES-NETO, R. F.; NAGANO, M. S. An iterated greedy approach to integrate production by multiple parallel machines and distribution by a single capacitated vehicle. **Swarm and evolutionary computation**, Elsevier, v. 44, p. 612–621, 2019.

TIMKOVSKY, V. Cycle shop scheduling. *In*: LEUNG, J. Y.-T. e. (ed.). **Handbook of scheduling – algorithms, models and performance**. Boca Ratton: Chapman & Hall, 2004. p. 111–132.

TOTH, P.; VIGO, D. **The vehicle routing problem**. [*S.l.: s.n.*]: Siam, 2002.

TROJET, M.; H'MIDA, F.; LOPEZ, P. Project scheduling under resource constraints: Application of the cumulative global constraint in a decision support framework. **Computers & Industrial Engineering**, Elsevier, v. 61, n. 2, p. 357–363, 2011.

TUBINO, D. F. **Planejamento e controle da produção: teoria e prática .** [*S.l.: s.n.*]: Editora Atlas SA, 2017.

ULLRICH, C. A. Integrated machine scheduling and vehicle routing with time windows. **European Journal of Operational Research**, Elsevier, v. 227, n. 1, p. 152–165, 2013.

VINCENT, F. Y. *et al.* Adaptive neighborhood simulated annealing for the heterogeneous fleet vehicle routing problem with multiple cross-docks. **Computers & Operations Research**, Elsevier, v. 129, p. 105205, 2021.

VINCENT, F. Y.; LIN, S.-W.; CHOU, S.-Y. The museum visitor routing problem. **Applied Mathematics and Computation**, Elsevier, v. 216, n. 3, p. 719–729, 2010.

WANG, D.-Y.; GRUNDER, O.; MOUDNI, A. E. Integrated scheduling of production and distribution operations: a review. **International Journal of Industrial and Systems Engineering**, Inderscience Publishers, v. 19, n. 1, p. 94–122, 2015.

WITKOWSKI, T.; ANTCZAK, P.; ANTCZAK, A. Hybrid method for solving flexible open shop scheduling problem with simulated annealing algorithm and multi-agent approach. *In*: TRANS TECH PUBL. **Advanced Materials Research**. [*S.l.: s.n.*], 2012. v. 383, p. 4612–4619.

WU, X.; CHE, A. Energy-efficient no-wait permutation flow shop scheduling by adaptive multi-objective variable neighborhood search. **Omega**, Elsevier, v. 94, p. 102117, 2020.

YAO, M.-J.; SOEWANDI, H.; ELMAGHRABY, S. E. Simple heuristics for the two machine openshop problem with blocking. **Journal of the Chinese Institute of Industrial Engineers**, Taylor & Francis, v. 17, n. 5, p. 537–547, 2000.

YUNUSOGLU, P.; YILDIZ, S. T. Constraint programming approach for multi-resource-constrained unrelated parallel machine scheduling problem with sequence-dependent setup times. **International Journal of Production Research**, Taylor & Francis, p. 1–18, 2021.

ZARANDI, M. F.; KHORSHIDIAN, H.; SHIRAZI, M. A. A constraint programming model for the scheduling of jit cross-docking systems with preemption. **Journal of Intelligent Manufacturing**, Springer, v. 27, n. 2, p. 297–313, 2016.

ZHANG, D. *et al.* A vehicle routing problem with distribution uncertainty in deadlines. **European Journal of Operational Research**, Elsevier, 2020.

ZHANG, J.; WANG, L.; XING, L. Large-scale medical examination scheduling technology based on intelligent optimization. **Journal of Combinatorial Optimization**, v. 37, n. 1, p. 385–404, jan. 2019. ISSN 1573-2886.

ZHUANG, Z. *et al.* A heuristic rule based on complex network for open shop scheduling problem with sequence-dependent setup times and delivery times. **IEEE Access**, Ieee, v. 7, p. 140946–140956, 2019.

ZHUANG, Z. *et al.* An improved artificial bee colony algorithm for solving open shop scheduling problem with two sequence-dependent setup times. **Procedia CIRP**, Elsevier, v. 83, p. 563–568, 2019.

ZOBOLAS, G.; TARANTILIS, C. D.; IOANNOU, G. Solving the open shop scheduling problem via a hybrid genetic-variable neighborhood search algorithm. **Cybernetics and Systems: An International Journal**, Taylor & Francis, v. 40, n. 4, p. 259–285, 2009.

APPENDIX

APPENDIX A – DATA REPOSITORY

The data management of the doctoral dissertation is based on the free availability of the contents proposed, aiming at the literature growth of the problem and the possibility of using the proposed solution methods in practical projects in industries and other areas with a production environment similar to the one studied in the research. In the data repository, we divided the main data generated into four categories:

- Source codes of the exact, heuristic, metaheuristic, and matheuristic methods developed in Python, Julia, and OPL languages.
- Instance sets used for methods comparison. Instances are available in `.txt` or `.csv` formats.
- Tables with the individual results of each method on each instance. Tables are available in Excel in `.xlsx` format.
- Results of the statistical tests performed, developed in R language, and presented in Excel in `.xlsx` format.

The instance sets, codes, figures, tables, and results do not involve ethical or legal issues. Therefore, they may be used in research projects, provided the source (doctoral thesis or articles of each thesis's chapter) is cited. They can also be reproduced in public documents, such as scientific articles and technical reports, as long as the source is cited. Table 44 provides the link to the data repository for each of the five chapters of the thesis:

Table 44 – Doctoral dissertation datasets

Problem	Data	Access link
Classic open shop	Experiments results, developed algorithms, and statistical tests	Link I
Open shop with setup times	Experiment results, problem instances, proposed algorithms, and statistical tests	Link II
Open shop with machine blocking	Experiment results, problem instances, proposed algorithms, and statistical tests	Link III
Open shop with repetitions of operation	Experiment results, problem instances, proposed algorithms, and statistical tests	Link IV
Open shop with vehicle routing	Experiment results, problem instances, and proposed algorithms	Link V

Source: Authors.

EESC • USP