Journal of Project Management 8 (2023) 99–118

Contents lists available at GrowingScience

Journal of Project Management

homepage: www.GrowingScience.com

Bounded dynamic programming approach to minimize makespan in the blocking flowshop problem with sequence dependent setup times

Edson Antonio Gonçalves de Souza[a], Marcelo Seido Nagano[a*], Hugo Hissashi Miyata[a], Levi Ribeiro de Abreu[a]

[a]*University of São Paulo, São Carlos School of Engineering, Av. Trabalhador São-Carlense, 400, 13566-590, São Carlos, São Paulo, Brazil*

C H R O N I C L E

Article history:
Received: September 1, 2022
Received in revised format: September 29, 2022
Accepted: November 29, 2022
Available online: December 1, 2022

Keywords:
Blocking Flowshop
Setup Times
Makespan
Bounded Dynamic Programming

A B S T R A C T

This paper aims at presenting an algorithm for solving the blocking flow shop problem with sequence dependent setup times (BFSP-SDST) with minimization of the makespan. In order to do so, we propose an adapted Bounded Dynamic Programming (BDP-SN) algorithm as solution method, since the problem itself does not present a significant number of sources in the state-of-art references and also because Dynamic Programming and its variants have been resurfacing in the flowshop literature. Therefore, we apply the modified method to two sets of problems and compare the results computationally and statistically for instances with a MILP and a B&B method for at most 20 jobs and 20 machines. The results show that BDP-SN is promising and outperforms both MILP and B&B within the established time limit. In addition, some suggestions are made in order to improve the method and employ it in parallel research regarding other branches of machine scheduling.

1. Introduction

The procedures involved in the manufacturing systems are multistage processes that require an integration in order to align the different levels that define the goals of a given company. According to Koren et al. (1999), due to the aggressive dynamics imposed by competitiveness and profitability, it is relevant to develop methods that can be easily and rapidly upgraded and optimized. In addition, due to the global changes that are focused on sustainability and more compact designs (e.g. Mahalik and Nambiar (2010), Seow and Rahimifard (2011)), innovation has become an essential pillar in order to enable the best trade-off between profitability and cleaner ways of launching a given product to the custumers. Due to these facts, the use of resources necessary to optimize numerous stages in the manufacturing process is one of the recurrent tools that companies have been applying recently (e.g. Hon, 2005; Lepuschitz et al., 2010) and one of these stages is known as machine scheduling.

Pinedo (2012) defines machine scheduling as the allocation of a set of machines to a set of jobs that are essential compounds for the final product manufacturing. These jobs must be ordered according to a given flow pattern inherent to the production system and technological constraints in order to optimize a given objective function of interest. Conventionally, some notations have been developed over the years so as to represent a scheduling problem according to the variants stated above, however the one that commonly appears in papers is that devised by Graham et al. (1979), also known as the three-field notation $\alpha|\beta|\gamma$. The α-field contains information about the flow pattern described by the problem. In case of multi-machine

* Corresponding author.
E-mail address: drnagano@usp.br (M. S. Nagano)

doi: 10.5267/j.jpm.2022.12.001

environments it also specifies the number of machines involved. The β-field is filled whenever a constraint or a set of constraints is a characteristic of the problem of interest. The γ-field is the last compound and it conveys to the reader the objective function or the set of objective functions that require optimization.

Regular functions have been the basis of machine scheduling and are still investigated by many authors in the state of art references using a variety of solution methods (e.g. Johnson (1954), Ignall and Schrage (1965), Moore (1968), Lawler and Moore (1969), Potts and Van Wassenhove (1985), Süer, Báez, and Czajkiewicz (1993)), however, with the advent of new techniques, regulations and technologies that have been inserted in the industrial scenario, machine scheduling problems have also been required to evolve and reshape themselves in order to aggregate these novel characteristics. This phenomenon caused the number of technological constraints and objective functions to increase by combining multiple constraints (e.g. Wang, Yin, and Liu (2016), Agnetis and Mosheiov (2017), Mor and Shapira (2020), Li and Yuan (2020)) and performance measures and therefore, generating a set of problems that are more compatible with the contemporary production framework.

Flowshop scheduling has been widely explored when compared to the other shop environments and this setting is commonly found in manufacturing scenarios. Among the several variants of flowshop, the one that is more recurrent in terms of research is the permutation flowshop problem (PFSP) and according to Maccarthy and Liu (1993) it can be defined as a serial configuration in which the order of jobs is constrained to be the same for all the machines. Since the first paper published regarding this topic (see Johnson (1954)), a substantial amount of articles have been produced and as systems admitted more complex characteristics, flowshop has also followed this trend. Gupta and Stafford Jr (2006) report the existence of at least 1200 ranging from 1954 to 2006 and certainly a significant number can be added up to this date, given that more contemporary approaches needed to be investigated and algorithms were required to optimize these systems.

Regarding the constraints, one that is commonly cited and investigated by authors is blocking, which is a technological constraint imposed usually on flow-shops whenever the system does not present an intermediate buffer. Therefore, if a subsequent machine is yet not available, the current job remains blocked on the current machine until it can be transferred to the next. Blocking flow-shop scheduling (BFSP) is a problem that often arises in several segments of manufacturing. Applications regarding its occurrence have been reported in iron and steel industries (see Gong, Tang, and Duin (2010)), industrial waste, metallic parts manufacturing (see Martinez et al. (2006)), chemical and pharmaceutical industries (see Merchan and Maravelias (2016)) and robotic cells (see Ribas et al., 2015), among others.

Another technological constraint that commonly emerges in the industrial context is the inclusion of activities in which setup times are added to the process execution. These times are relevant because they represent the preparation for receiving a given job and, since it may be a significant amount in the manufacturing environment, they might as well be added. It is important to mention that regarding setup times, two of them are extremely important in literature and they are classified according to their dependence on the sequence. Sequence independent setup times are those that depend only on the job that will be allocated and is usually denoted by s_{jk}, while sequence dependent setup times depend either on the jobs that have been previously allocated and the one that will be placed next, being usually denoted by s_{ijk}. Some other variants also exist, however they are not relevant to the scope of this research and their definition will not be given. Allahverdi et al. (2008) collected over 300 articles regarding the subject (also including setup costs) and demonstrated that, throughout the years, not only have setup times drawn the attention of many researchers but also it has helped develop knowledge for a whole new set of problems that are often recurrent in industry (e.g. job family setup times). Sequentially, in a third comprehensive article on the subject (see Allahverdi (2015)), the author gathers information on about 500 papers and shows that setup costs/times are a problem that indeed present potential in practical and theoretical aspects.

Nevertheless, some of the combinations of blocking and setup times are frequently neglected in research, even though they might have substantial participation in realistic manufacturing systems. For instance, in a recent review paper, Miyata and Nagano (2019) showed that, among 140 papers regarding the blocking flow-shop, only six of them would reckon setup times in single objective functions. It can be evidenced that some solutions for such problem have been relying on metaheuristics (e.g., Shao, Pi, and Shao (2018)), which have produced outstanding results for large-scale jobs for Taillard benchmark, constructive heuristics, that can be found in Takano and Nagano (2019), which showed that a profile fitting heuristic adapted to the setup times, whether combined with NEH heuristic or not, was the most efficient one compared to other 13 heuristics. Newton et al. (2019) developed an acceleration method and a constraint-guided local search in order to solve the mixed blocking flow-shop problem, which combines RCb (Release when completing blocking) and RSb (Release when starting blocking) conditions and outperformed the other heuristics and metaheuristics previously developed by adapting them to the mixed problem. Apart from that fact, a MILP and a B&B have been proposed by Takano and Nagano (2017) as exact methods and instances with up to 20 jobs and a combination of machines that would be considered 10 machines. Additionally, it is relevant to mention that all these papers have focused on minimizing the makespan. Hence, one can notice that just few methods have been designed to solve this class of problems, showing that this is an uncharted area in scheduling with potential for research, mostly for exact methods or algorithms based on exact methods.

Therefore, this paper aims at presenting an alternative solution so as to solve the blocking flowshop problem with sequence dependent setup times (BFSP-SDST) with minimization of makespan $Fm|s_{ijk}, block|C_{\max}$. In this version of the problem,

we considered anticipatory setup times, i.e. the setup of a given machine to start processing a new job is allowed to start immediately after the previous job leaves such a machine, as it is shown in Fig. 1. Here, we adapt the Bounded Dynamic Programming (BDP) method proposed by Joaquín Bautista et al. (2012) for the BFSP to the BFSP-SDST. Despite not being a recent model, BDP has been resurfacing and has been applied to the shop environments, showing substantial results (Joaquín Bautista et al., 2012; Ozolins, 2018; Ozolins, 2019a; Ozolins, 2019b). In order to compare the efficiency of the algorithm proposed in this paper (BDP-SN), we coded a mixed integer linear programming (MILP) algorithm and the branch and bound (B&B) proposed by Takano and Nagano (2017) and divided the problems into two sets. The first set compares the performance of the BDP-SN with MILP and B&B and the set of instances used are the same generated in Takano and Nagano (2017). The second set only compares the performance of BDP-SN with MILP with a brand new set of instances generated randomly according to an uniform distribution. In both cases, BDP-SN outperforms the MILP and B&B methods in a general analysis, considering at most 20 jobs and 20 machines. Although we aforementioned some heuristics and metaheuristics to solve this particular problem in scheduling, the motivation for this paper is to introduce dynamic programming-based features for solving it, since this form of solution has not yet been proposed for the BFSP-SDST and therefore, only comparisons with exact methods are tested.

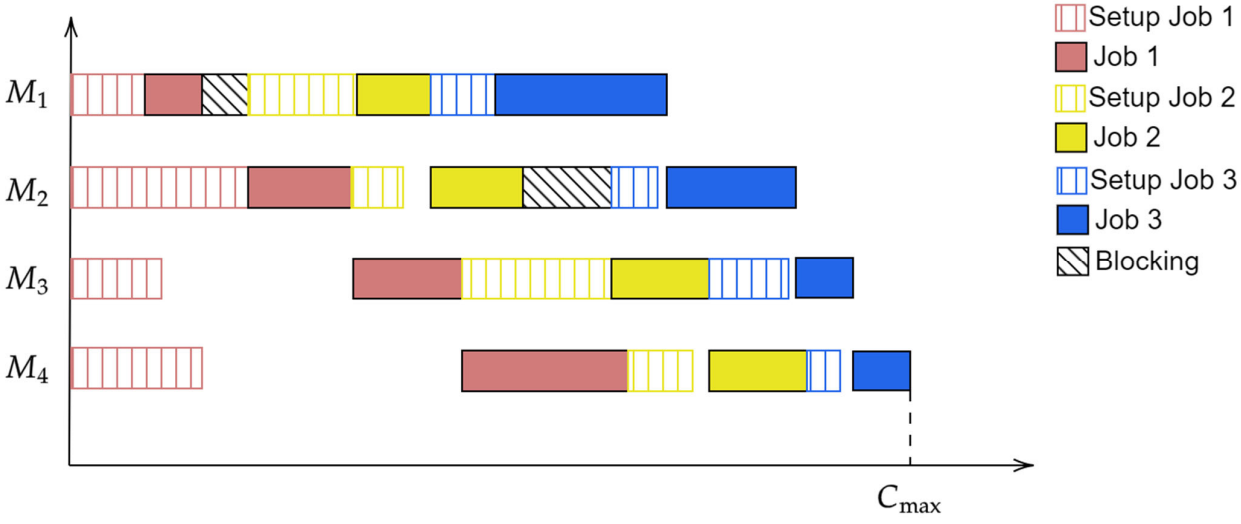

Fig. 1. Pictorial representation of the $Fm|s_{ijk}, block|C_{\max}$ for 3 jobs and 4 machines.

This paper is structured as follows. Section 2 delineates the modelling of the BFSP-SDST with notations necessary to understand the mathematical formulations. Section 3 presents the structure of the BDP and Section 4 extends it to BDP-SN to solve the BFSP-SDST. Section 5 encompasses the methodological part of the paper by detailing how the experiments have been carried out as well as the statistical measurements and tests that have been employed in order to establish the comparison among methods. Section 6 shows the results for all the methods that have been coded for BFSP-SDST. Section 7 develops the concluding remarks about this research and some suggestions that might be incorporated in order to enhance the BDP-SN algorithm and applications for further studies.

2. Problem statement and notations

Let $j \in J$ be a given job to be processed on a given machine $k \in M$ in a serial fashion, such that the sequencing of jobs must be exactly the same in every machine. In addition, let p_{jk} denote the time taken by job j to be processed on machine k and consider a setup time s_{ijk} between the preceding job $i \in J$ and j to indicate that a preparation must take place before job j starts its processing. For this case, the setup times are classified as anticipatory, i.e., a given machine k is allowed to be prepared while machine $k-1$ is still processing job j. Furthermore, whether a given job i is yet to finish its processing on machine $k+1$ or setup is being carried out and job j has already completed its processing on machine k, the latter must be held until machine $k+1$ is ready to begin its processing. This effect is known as blocking. D_{jk} represents the departure time, i.e. the point in time job j leaves machine k. Lastly, let $C_{\max}$ denote the maximum completion time for the last job n on the last machine k, i.e. makespan.

The problem described above is defined as $Fm|s_{ijk}, block|C_{\max}$ and aims at finding the optimal sequence for n jobs that minimizes $C_{\max}$. It can be modelled as follows:

$$\min C_{\max} = D_{[n]m} \tag{1}$$

$$D_{[1]k} = \max\{s_{[0][1]k+1}, D_{[1]k-1} + p_{[1]k}\}; \quad k = 2, \dots, m-1 \tag{2}$$

$$D_{j1} = \max\{D_{i2} + s_{ij2}, D_{i1} + s_{ij1} + p_{j1}\}; \quad j = 1, \dots, n \tag{3}$$

$$D_{jk} = \max\{D_{i(k+1)} + s_{ij(k+1)}, D_{j(k-1)} + p_{jk}\}; \quad i \neq j, \forall k \neq 1, m \tag{4}$$

$$D_{jm} = D_{j(m-1)} + p_{jm}; \quad \forall j \in J. \tag{5}$$

Eq. (1) refers to the minimization of the makespan. Constraint (2) refers to the allocation of the first job ($[l]$ stands for the job occupying the l-th position) for every machine, except for the first one and constraint (3) sets the departure time relative to each job scheduled in the first machine. Note that, since i precedes j, when j is set as the first job, i would be the zeroth job, and because it does not exist, $D_{i1} = 0$. Constraint (4) is the general recursive relation to obtain the departure times on every machine except for the last one, which, in turn, is calculated via constraint (5).

2.1 Mixed Integer Linear Programming (MILP)

As it has been previously stated, a MILP formulation has been used in order to establish a comparison with the method that is the core of this paper. Therefore, for sake of enlightenment, we present the mathematical formulation of the method, which has been developed in Takano and Nagano (2017), as follows.

Let $st_{[t]k}$ be the time a given job in position t starts its processing one machine k. Furthermore, let $z_{[t]ij}$ and $x_{[t]j}$ be binary variables that represent adjacency between job i and job j when the latter is inserted in position t and the assignment of job j in postion t, respectively. Then, the MILP formulation is given by:

$$\min C[n]m \tag{6}$$

subject to

$$\sum_{t=1}^{n} x_{[t]j} = 1; \quad j = 1, \dots, n \tag{7}$$

$$\sum_{j=1}^{n} x_{[t]j} = 1; \quad t = 1, \dots, n \tag{8}$$

$$\sum_{i=1}^{n} \sum_{i \neq j} z_{[0]ij} = 0 \tag{9}$$

$$\sum_{i=1}^{n} \sum_{i \neq j} z_{[t]ij} = 0; \quad t = 2, \dots, n \tag{10}$$

$$z_{[t]ij} \geq x_{[t-1]i} + x_{[t]j} - 1; \quad i = 1, \dots, n; j = 1, \dots, n; t = 2, \dots, n; i \neq j \tag{11}$$

$$st_{[1]1} = \sum_{j=1}^{n} x_{[1]j}\, s_{[0][1]1} \tag{12}$$

$$st_{[1]k} \geq \sum_{j=1}^{n} x_{[1]j}\, s_{[0][1]k}; \quad k = 2, \dots, m \tag{13}$$

$$st_{[1]k} \geq C_{[1](k-1)}; \quad k = 2, \dots, m \tag{14}$$

$$st_{[t]k} \geq st_{[t-1](k+1)} + \sum_{i=1}^{n} \sum_{i \neq j} z_{[t]ij} s_{ijk}\,; \quad k = 2, \dots, m-1; t = 2, \dots, n \tag{15}$$

$$st_{[t]k} \geq C_{[t]k}; \quad k = 2, \dots, m-1; t = 2, \dots, n \tag{16}$$

$$C_{[t]k} = st_{[t]k} + \sum_{j=1}^{n} x_{[t]j}\, p_{jk}; \quad k = 1, \dots, m; t = 1, \dots, n \tag{17}$$

$$x_{[t]j}, z_{[t]ij} \in \{0,1\}; \quad i, j = 1, \dots, n; t = 1, \dots, n; i \neq j \tag{18}$$

$$C_{[t]k}, st_{[t]k} \geq 0; \quad k = 1, \dots, m; t = 1, \dots, n. \tag{19}$$

Equation (6) denotes the minimization of makespan. Constraint (7) and constraint (8) guarantee that each job is only assigned to a single position and that the opposite also holds. Constraint (9) ensures that the binary variable relative to the setup times is zero, since no $j \in J$ must occupy the zeroth position. Constraint (10) shows that the setup times binary variable is only activated when the job i in the $(t-1)-th$ position precedes job j in the $t-th$ position. Constraint (11) may be valued as -1, 0 or 1, if neither job i nor job j are assigned to the $t-1$ and t positions, if only one of the jobs is assigned to the correct position or both of them are correctly assigned to their positions, respectively. Constraint (12) calculates the start time of the first job on the first machine. Constraint (13) and constraint (14) make sure that the start time of setup operations are executed after blocking or completion time of a given job. Constraint (15) and constraint (16) play the same role for every job, except the first one assigned. Constraint (17) computes the completion times of every job on every machine. Finally, Constraint (18) and constraint (19) describe the domain of the decision variables.

2.2 Bounded Dynamic Programming to the $Fm|block|C_{max}$ problem

Although several methods have been designed for solving a variety of problems in machine scheduling, one may notice that the amount of heuristics and metaheuristics that researchers resort to in the present days is, by far, greater than the ones that are produced by exact methods. Clearly, the advantage of heuristics and metaheuristics relies on the fact that these methods are able to obtain good quality solutions in a very small CPU time and storage requirements are usually shorter that the ones observed in enumeration methods, for instance. However, these methods might also be tentative, since their functioning might depend on other heuristics or metaheuristics that, in turn, might not result in solutions as satisfactory as one could hope. Furthermore, another factor that is also relevant is that improvements on exact formulations usually generate optimality properties that might help describe a whole set of problems and classify them by providing exact-based algorithms, pseudo-polynomial and approximation algorithms, thus enlarging the set of useful mechanisms to solve problems of a more realistic framework.

It is known that pure DP algorithms, despite being more efficient than an explicit enumeration method, also are associated with storage problems due to its recursive structure. Hence, recent papers in machine scheduling have been leaning towards hybrid or derived methods from DP, which seem to be able to yield high quality solutions for a significant number of instances at less demanding conditions in terms of computational requirements. Although the contribution of DP to the permutation flowshop is not as extensive as it is for less complex environments, some work has been developed in order to verify the behaviour of such a method when applied to such an environment. On a smaller scale, some authors have derived methods from DP so as to solve the blocking flowshop variation and the results have also been satisfactory, indicating that some could be improved and used in further studies regarding the topic and branches of it. Among these methods, one that has been employed in shop environments is the Bounded Dynamic Programming (BDP).

According to Joaquin Bautista and Pereira (2009), BDP is an exact algorithm that combines features of DP and B&B that, under certain circumstances, can be employed as a heuristic. One of the advantages of such a method is that lower bounds and either heuristic or exact dominance rules are embedded in its structure so as to reduce the state space intrinsic to the problem, thus allowing results that would possibly not be achieved by pure DP in an acceptable time or computational requirements. It is worth mentioning that DP-based approaches are defined as a graph-like structure $G(L,T)$, where L represents the states generated once a given vertex begins branching and T is referred to the decision making process of transforming one state into another, which is denominated transition. Consequently, the problem is solved once there are no further transitions to take place and the graph constitutes a path. Hence, it is equivalent to solving a shortest path problem.

BDP generally consists of four elements as follows:

- Generation of state space: This step is characterized by an initial stage $l = 0$, which represents an empty set and a subsequent stage $l = 1$ containing n states, each relative to the assignment of a given task. Computationally, these stages are created as two consecutive lists, one for each stage specifically. These lists are currently updated as the first list is emptied and the second is filled with viable policies;

- Dominance rule: Artifice in which two states that belong to the same stage are compared and the one that presents the largest-valued policy for a partial sequence is dominated, thus being removed from the current stage;

- Bounding scheme: Lower bounds are introduced and calculated for each state. Whenever $LB \geq UB$ the state is eliminated from the current stage;

- Window width: This parameter, which is denoted by H, is used as a threshold for the number of states generated at a given stage and therefore, does not allow memory overreach due to the curse of dimensionality of a pure DP approach.

In order to give a detailed explanation on how these parts are assembled, we discuss the algorithm proposed by Joaquín Bautista et al. (2012) for the $Fm|block|C_{\max}$. Each list contains a vertex denoted by $X(j,l)$, which is equivalent to a given

state at stage l when allocating a given job j. In addition, a vertex is composed of three elements, such that $X(j,l) = [\vec{q}(j,l); \vec{D}(j,l); C_{\max}]$, where $\vec{q}(j,l)$ is defined as a binary vector for the assignment of job j, $\vec{D}(j,l)$ denotes the vector of departure times and $C_{\max}$ corresponds to the makespan of the a partial sequence at stage l. Note that $\vec{q}(j,l)$ is an actual representation of a partial sequence and therefore:

$$\sum_{p=1}^{n} q_p(j,l) = l \tag{20}$$

where $q_p(j,l)$ represents one of the binary variables of vector $\vec{q}(j,l)$, indicating whether the p-th job has been placed up to stage $l \in L$.

The reason to define these components are that they are strictly related to the dominance rule established for reducing the state space. Consider j and j' jobs to be allocated at a given stage l. Then it leads to:

$$X(j,l) \prec X(j',l) \Leftrightarrow [\vec{q}(j,l) = \vec{q}(j',l)] \wedge [\vec{D}(j,l) < \vec{D}(j',l)] \tag{21}$$

and

$$X(j,l) \equiv X(j',l) \Leftrightarrow [\vec{q}(j,l) = \vec{q}(j',l)] \wedge [\vec{D}(j,l) = \vec{D}(j',l)]. \tag{22}$$

Equation (21) refers to a situation where vertex $X(j,l)$ dominates $X(j',l)$ since they contain the same jobs allocated in the partial sequence, however, the former yields a smaller vector of departure times, which means that at least one entry of $D(j,l) < D(j',l)$. and for the remaining entries $D(j,l) \leq D(j',l)$ Similarly Equation (22) stands for an equivalence of vertices since the vector of departure times is strictly equal.

By applying these relations, a smaller state space is already being built, since when comparing vertices, the most promising one will undergo transition to the next stage while the other will be removed, thus avoiding a complete enumeration of all sequences. In Fig. 2, we can observe that the transition from one stage to another occurs by branching one of the nodes. The choice is made via dominance rule and therefore $X(j',l)$ is removed once it is assumed that $\vec{D}(j,l) < \vec{D}(j',l)$.

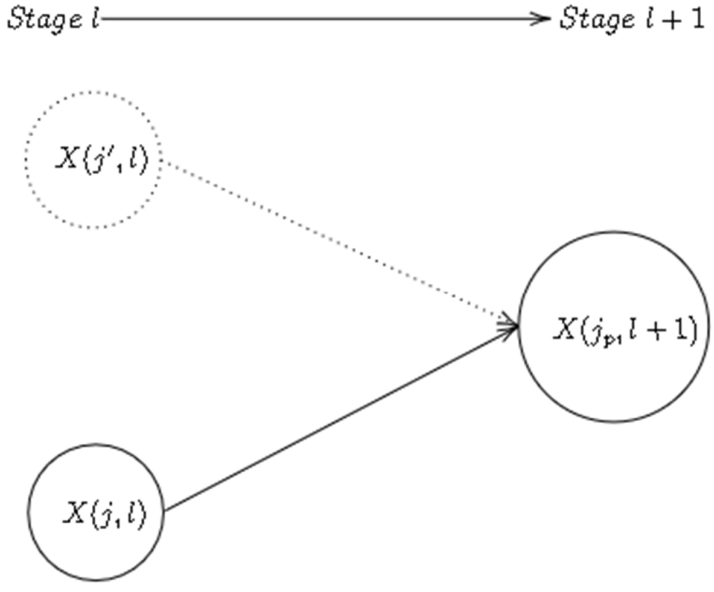

Fig. 2. Schematics of dominance rule and transitional state in the graph structure.

Although this rather "simple" configuration, in fact, diminishes the state space involved and delivers a shrunk graph structure, for practical purposes, this might still not be as efficient as it is expected from an improved method. Therefore, the authors also devise a job-based lower bound and a machine-based lower bound. Since the latter attains better solutions, the one we focus on is formulated as follows:

$$LB_k(j,l) = D_k(j,l) + \sum_{i \in \overline{S}} p_{ik} + \min_{i \in \overline{S}} \left\{ \sum_{h=k+1}^{m} p_{ih} \right\}, \quad k = 1, \ldots, m-1 \tag{23}$$

$$LB_k(j,l) = D_k(j,l) + \sum_{i \in \overline{S}} p_{ik}, \quad k = m \tag{24}$$

and, consequently, the lower bound for the vertex, is given by:

$$LB(j,l) = \max_{k \in M} \{LB_k(j,l)\}. \tag{25}$$

where $\overline{S}$ denotes the set of unscheduled jobs at a given stage l, $LB_k(j,l)$ represents the lower bound for a given machine and $D_k(j,l)$ the departure time of job j at machine k.

Feasible policies in BDP are commonly bounded by an upper bound and a lower bound. The latter is frequently obtained by some auxiliary method such as heuristics in the form of an initial solution. If the $LB \geq UB$ for a given vertex, then it can be disregarded, otherwise the algorithm will allow solutions whose values are worse than the initial one.

Before considering the window width parameter, one must be aware that in order to soothe the computational burden of seeking vertices randomly for comparison when using the dominance rule, it is recommended to sort the vertices. This sorting is carried out taking into account a priority rule vector $(LB(j,l); C_{\max})$, which can follow two variants. The first one sorts the vertices in an increasing order of $LB(j,l)$ and, in case ties occur, the $C_{\max}$ is selected to determine the sorting. The second variant operates in an analogous manner by shifting the $C_{\max}$ and $LB(j,l)$.

Ultimately, the window width works as a regulator for the scheduler. The bigger its size, the larger the number of solutions that can be admitted in the state space, i.e. the number of feasible solutions at a given stage l is bounded by H. Although it is a very effective measure in terms of computational effort, the drawback of utilizing is that it may transform the exact approach into a heuristic. It can be explained by the fact that an intermediate vertex of the graph, one which provides the shortest path, may be discarded, as shown in Fig. 3.

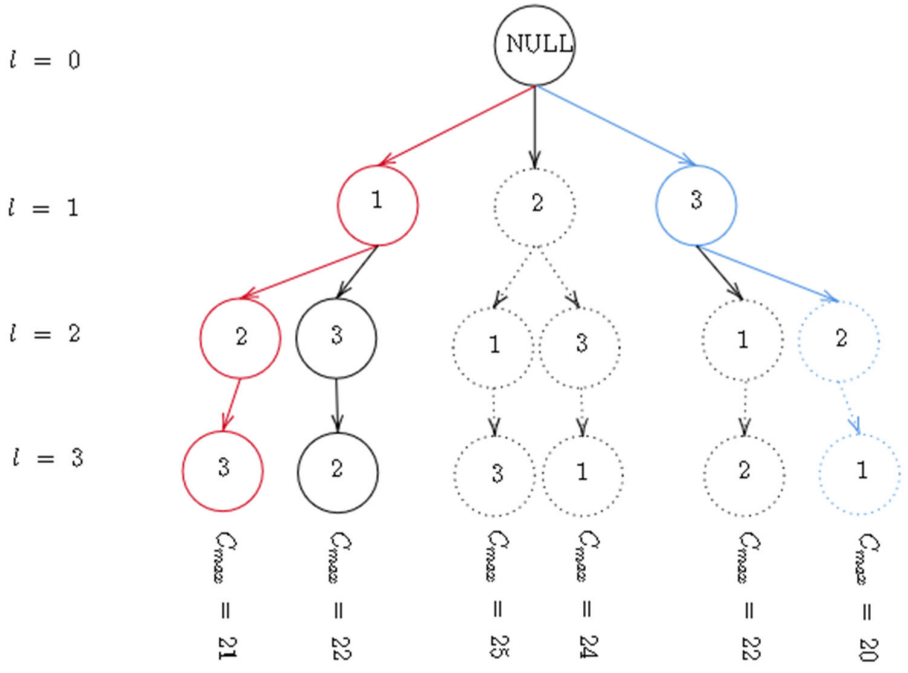

Fig. 3. Representation of BDP and its perspective as a heuristic procedure considering the criteria for space state reduction.

A remark on this parameter is also the fact that it has an empirical nature and despite being shown in previous papers, the definition for values depends on the problem and the computational limitations for a particular experiment.

The authors identify the exact behavior of the BDP if the following two conditions are satisfied:

$$\max_{1\le l\le L}\{H(l)\} < H \tag{26}$$

$$(\max_{1\le l\le L}\{H(l)\} = H) \wedge (D_m(j,L) < LBZ_{\min}) \tag{27}$$

where represents the lowest LB among all the vertices that have been disregarded throughout the process. In case these conditions are not satisfied, the algorithm will present a heuristic behavior. Note that, since only two lists are allowed for storage, the first list should be empty when all the jobs in it have been branched and the feasible policies remain stored in the second list. Thus, in order to reuse them, the vertices in the second list are transferred to the first one, stage is updated to become stage and, once again the process of branching starts in order to fill the second list, which is now empty.

Once the algorithm's profile has been outlined for this specific problem, the pseudo-code will be presented only in the next subsection, however a final remark should be made about BDP. One can notice due to the similarities to the B&B and also Beam Search methods, some misconceptions might occur regarding the functioning of BDP, so a simple description on the differences might be needed. First, when comparing the BDP and B&B (with Breadth First Search algorithm, for example), the BDP relies on a graph rather than a tree to perform the search of the optimal shortest path, which in fact, makes it more simple. Also, the B&B passes through every possible state in the tree, whereas this is not required in the BDP. When compared to the Beam search method, despite the fact that either method contains a window width in its formulation, BDP makes mandatory use of bounds, while the Beam Search does not require them. Additionally, modern approaches of Beam Search have included some randomness in their formulations, whereas BDP, as of date, is purely deterministic. Moreover, Beam Search also relies on a tree-based structure.

2.3 BDP extended for the $Fm|s_{ijk}, block|C_{max}$ problem

This section is not supposed to be extensive because the major portion of the algorithm's functioning has been detailed in the previous section. Hereafter, the adaptation will be carried out taking into account two factors: slight modifications in the BDP algorithm proposed by Joaquín Bautista et al. (2012) and a major one, which is providing a lower bound that performs in an advantageous manner, i.e. finding the closest vector to $\vec{v} = (0,0)^T$ in the Pareto frontier analysis regarding solution quality and computational effort.

As stated in Tomazella and Nagano (2020) recent review on B&B, problems regarding the permutation flow-shop scheduling with sequence dependent setup times are frequently solved via heuristics and metaheuristics, once these methods obtain high quality solutions within a small time frame. In case of a B&B approach for the same problem, the tightness of the bound is a crucial factor in determining the performance of the algorithm and it becomes even more costly when setup times are included because they are considered the most complex part of the LB formulation. Therefore, either improvements on lower bounds formulations are often required or their employment might be more adequate in other methods.

Takano and Nagano (2017) proposed a B&B method by designing four lower bounds for solving the $Fm|s_{ijk}, block|C_{\max}$ problem. Compared to bounds previously proposed for the $Fm|s_{ijk}|C_{\max}$ and $Fm|block|C_{\max}$ separately, these bounds can be gauged to a less complex structure and not being a tight bound, despite calculating solutions more quickly, may generally demand more time to achieve high quality solutions. Nevertheless, the authors present an upper bound formulation for the idle times ($\delta_j^{(k)}$) and a lower bound for the blocking ($\mathrm{LBB}_j^{(k)}$), which are relatively simple to be computed and given respectively by:

$$\delta_j^{(k)} = \max\{0, (\delta_j^{(k-1)} + s_{ij(k-1)} + p_{j(k-1)}) - (s_{ijk} + p_{ik})\} \quad \forall i \neq j \tag{28}$$

with

$$\delta_j^{(1)} = 0 \tag{29}$$

and

$$\mathrm{LBB}_j^{(k)} = \max\{0, (s_{ij(k+1)} + p_{i(k+1)}) - (\delta_j^{(k)} + s_{ijk} + p_{jk})\} \quad \forall i \neq j \tag{30}$$

with

$$\mathrm{LBB}_j^{(m)} = 0 \tag{31}$$

Unless the sequence to which the makespan obtained is previously known, the idle times and the blocking time cannot be calculated due to the fact the next scheduled job is a combinatorial problem. Therefore, due to some equivalences, it is possible to obtain the equations above regarding bounded values. Additionally, their initial conditions are overt because there exists no blocking on the last machine and no idle time on the first machine.

When comparing the structure of the LB proposed by Joaquín Bautista et al. (2012) and the one designed by Takano and Nagano (2017), it can be seen that they differ by including terms relative to the setup and the lower bound for blocking. Also, since the former yielded such promising results in the analysis for the $Fm|block|C_{\max}$ using BDP, it seems fitting to apply it to the variant where setup times are included. Additionally, once the efficiency of BDP is directly connected to the strength of the lower bound, selecting a bound that presents similarities with one that has already been successful for the BFSP may induce a better adherence of such LB to the BDP method by including alterations in its structure that are relative to the setup times. Some of the modifications depend on two other variables, which are the sum of minimum setup times (SMST) and sum of the minimum lower bounds for blocking (SMLBB) and they are defined as follows:

$$\mathrm{SMST}_k(j,l) = \sum_{\substack{u\in(\overline{S}\oplus\{j\})\\ v\in\overline{S}\\ u\neq v}} (\min(s_{uvk})) - \max_{\substack{u\in\overline{S}\\ v\in\overline{S}\\ u\neq v}} (\min(s_{uvk})) \tag{32}$$

$$\mathrm{SMLBB}_k(j,l) = \sum_{u\in(\overline{S}\oplus\{j\})} (\min(\mathrm{LBB}_u^{(k)})) - \max_{u\in\overline{S}} (\min(\mathrm{LBB}_u^{(k)})) \tag{33}$$

where S stands for the set of scheduled jobs and $\overline{S}$ the set of unscheduled jobs. The notation $u \in (\overline{S} \oplus \{j\})$ indicates that job u can be represented by any job that has not been yet scheduled, however job j is also considered a candidate. An additional remark that should be made about the computation in Equation (33) is that the second term refers only to unscheduled jobs. Therefore, when coupling the jobs in Equation (30) and Equation (28), one must be aware that both i and j for that set belong to $\overline{S}$.

Finally, the lower bound is given by:

$$\begin{aligned}\mathrm{LB}_k(j,l) \quad &= D_k(j,l) + \mathrm{SMST}_k(j,l) + \sum_{u\in\overline{S}} (p_{uk}) + \mathrm{SMLBB}_k(j,l) \\ &+ \min_{u\in\overline{S}} \left(\sum_{h=k+1}^{m} (p_{uh}) + \min\left(\sum_{h=k+1}^{m} (\mathrm{LBB}_u^{(h)}) \right) \right), \quad \forall k \neq m.\end{aligned} \tag{34}$$

and

$$\mathrm{LB}_m(j,l) = D_m(j,l) + \mathrm{SMST}_m(j,l) + \sum_{u\in\overline{S}} (p_{um}) \tag{35}$$

This LB as well as the others that are not part of the scope of this research have been adapted from the one proposed by Ronconi (2005) for the $Fm|block|C_{\max}$. The author based the LB construction on a graph representation of the problem by finding the path that minimizes the distance between the first and last vertices. In addition, one can also perceive similarities between this LB and the one seen in the previous subsection, which ensures that the latter, under adaptations such as the one presented in this subsection, is appropriate.

Finally, we introduce the pseudo-code for the BDP approach for the $Fm|s_{ijk}, block|C_{\max}$ in Fig. 4. The algorithm has as Input: $n, m, H, p_{jk}, s_{ijk}, UB$ and as Output: $C_{max}, List(L)\ of\ optimal\ sequences$. Note that in order to consider the lower bound as a reduction method for the state space of the algorithm, an upper bound must be also calculated. The aforementioned sources that applied BDP to a given problem suggest either the use of heuristics such as Greedy procedures/local search or BDP with a small window width (e.g., $H = 1$).

```
Initialization: l = 0, LBZ_min = ∞, S = ∅
while l < n do
    l ← (l + 1)
    if l = 1 then
        forall j ∈ S̄ do
            if LB(j, l) ≥ UB then
                delete X(j, l)
                S ← S
            else
                S ← (S ⊕ {j})
                if |List(l)| > H then
                    insert X(j, l) in List(l)
                    sort List(l) according to (LB; C_max) priority
                    remove last vertex in List(l)
                else
                    insert X(j, l) in List(l)
                    sort List(l) according to (LB; C_max) priority
    else
        while |List(l)| ≠ 0 do
            select the first vertex X(i, l) in List(l)
            forall j ∈ S̄ do
                if LB(j, l + 1) ≥ UB then
                    delete X(j, l + 1)
                    S ← S
                else
                    forall X(j′, l + 1) do
                        if X(j′, l + 1) ≺ X(j, l + 1) then
                            delete X(j, l + 1)
                        else
                            S ← (S ⊕ {j})
                            if |List(l + 1)| > H then
                                insert X(j, l + 1) in List(l + 1)
                                sort List(l + 1) according to (LB; C_max) priority
                                remove last vertex in List(l + 1)
                            else
                                insert X(j, l + 1) in List(l + 1)
                                sort List(l + 1) according to (LB; C_max) priority
            remove X(i, l) from List(l) and make next vertex first vertex
        while |List(l + 1)| ≠ 0 do
            List(l) ← List(l + 1)
```

Fig. 4. BDP algorithm for the $Fm|s_{ijk}, block|C_{\max}$

Note that this algorithm is composed of two main parts. The first one is related to the initial allocation of jobs. First, a null set is programmed to be the first list. When the first jobs are branched into vertices, the second list is created to store them. For them the fathoming can occur only via the upper bound analysis or the window width parameter. The list is updated according to these criteria and becomes the first list by emptying the second one. For the remaining levels, the process is similar, however the comparison among the permutations with the same allocated jobs is considered to reduce the state space.

3. Computational Experiments

The computational analysis is divided into two sets of experiments and each set will be characterized by a job set, a machine set, a time limit, the distribution of processing times and the distribution set of setup times. Also, for the sake of organization, two subsections are created in order to introduce methods used for comparison as well as to establish a proper configuration of each set. Additionally, before introducing the description of each set, it is relevant to mention that the BDP and B&B methods have been coded in C language in a PC Intel(R) Core(TM) i5-2500K CPU 3.30 GHz with 4 GB RAM under a 64 bits Windows 7 operating system.

3.1 First set of problems

The first set of problems is based on the one carried out in Takano and Nagano (2017) by considering 540 problems and subdividing them into 27 classes of 20 instances each, where a given class is defined by a combination of machines being assigned to a given number of jobs. For the first 18 classes, the job set is defined as $n = \{10,12,14\}$ and the machine set is defined by $m = \{2,3,4,5,7,10\}$. For the remaining classes, the combination is given by $n = \{16,18,20\}$ and $m = \{2,3,4\}$. In addition, each instance will be terminated when the time limit of 3,600 seconds is reached.

Regarding the processing times the authors use the data provided by Ronconi (2005) considering a uniform distribution for values in a $[1,100]$ interval. As for the setup times, the interval also is uniformly distributed in a $[1,100]$ interval. Therefore, in this first analysis, the database will be the same used by these authors.

The BDP algorithm will be a replicated experiment by varying only the value of the window width, whose set is defined as $H = \{1,10,50,100,250,500,750,1000,1250,2500,5000,10000,20000\}$, thus each replication will be treated as an independent method. In addition, for a given H, the value of the upper bound UB for the current problem, is given by the value of $C_{\max}$ obtained for the same instance in the preceding window width. For $H = 1$, the UB value is set to infinity, given that no preceding window width exists.

The comparisons for this experimental set rely on the results obtained by programming the BDP and comparing its results with a MILP procedure and a B&B method, which has also been provided by Takano and Nagano (2017). It is important to mention that the MILP has been coded in Python by using Python 3.6.5 interface and run by the IBM(R) ILOG CPLEX 12.8 solver. Although the functioning of the MILP has been carefully described and its mathematical formalism has already been presented in section 2, the reader might question the reason the same has not been done for the B&B method. Clearly, the strength of the B&B is related to the quality of the LB defined (which might be its most relevant quality) and since it would be already presented in section 3 for our BDP variants, we did not deem necessary the description of the B&B itself.

3.2 Second set of problems

This set of problems contains 2880 problems with a job set denoted by $n = \{4,5,6,7,8,10,12,15,20\}$ and a machine set defined by $m = \{2,3,4,5,7,10,15,20\}$. Therefore there are 72 classes containing 40 instances each, which are internally divided into four classes of 10 instances each, according to the setup time classification that will be explained briefly ahead. Also, to maintain the equality of comparison to other methods, the time limit here for every experiment will be set to 3,600s.

The processing times for each problem are randomly generated according to an uniform distribution considering the interval for the data to be $[1,100]$. Regarding the setup times, four classes have been defined and they are denoted as $ST_{(25)}$, $ST_{(50)}$, $ST_{(100)}$ and $ST_{(125)}$. The number within the parentheses represents the maximum value assumed by the setup times data at a given interval, which means that the first class considers setup times in $[1,25]$, the second one in $[1,50]$, the third one in $[1,100]$ and the last one in $[1,125]$.

The algorithms compared will be the BDP and the MILP with a slight modification from the BDP applied in the previous experimental set. Once this development has been designed for a larger set of machines, the window width will be evaluated with a maximum value of 10,000. Besides that consideration, the premises exhibited in the previous subsection still hold.

3.3 Statistical Analysis

This subsection is a direct ramification from the previous one because the measurements related to the performance of the proposed model and the ones used for comparison are only verified through statistical evaluation and the three main datasets that can be extracted from the computational analysis are the makespan obtained for each algorithm, the CPU time and the number of nodes generated. Although this data is furnished, it is necessary to transform it into viable information. Therefore, two variables that will be constantly used in our analysis will be the relative percentage deviation (RPD) and the average relative percentage time (ARPT).

The RPD computes the percentage deviation of the solution for a given problem when compared with the best result found among the methods for this same problem. Let ρ denote a given problem, β be a given method among all the B methods used for comparison and $C^*_{\max}$ be the best makespan among all the methods, i.e. $C^*_{\max}(\rho) = \min_{1\leq\beta\leq B}\{C_{\max}(\rho,\beta)\}$. Then:

$$RPD(\rho,\beta) = 100\left(\frac{C_{\max}(\rho,\beta) - C^*_{\max}(\rho)}{C^*_{\max}(\rho)}\right). \tag{36}$$

The RPD measures the solution quality of the method proposed compared to others. Therefore, the closer to zero, the better the solution found by a given method, since there exists a small difference between the makespan for a method and the best one reported.

The ARPT functioning, which is a method attributed to Fernandez-Viagas, Leisten, and Framinan (2016) can be similarly described, however some additional calculations need to be performed in order to reach the final value. Let $T(\rho,\beta)$ be the CPU time for a given problem ρ using a given method β. Then:

$$AT(\rho) = \frac{\sum_{\beta=1}^{B} T(\rho,\beta)}{B} \tag{37}$$

which can be successfully applied to find the RPT given by:

$$RPT(\rho,\beta) = \frac{T(\rho,\beta) - AT(\rho)}{AT(\rho)} \tag{38}$$

and yield the ARPT defined by:

$$ARPT(\beta) = \frac{\sum_{\rho=1}^{R} RPT(\rho,\beta)}{R}. \tag{39}$$

These two deviation measures are relevant for the two types of experimental sets since they will be used for the most part of the analysis. A third measurement that can be used is the success rate of the algorithm, which estimates the frequency attributed to a given method in finding the best solution among all the methods. Thus, it can be defined as:

$$SR(\beta) = \frac{\sum_{\rho=1}^{R} G_{\rho,\beta}}{R} \tag{40}$$

where $G_{\rho,\beta}$ is a binary variable that is set to 1 if the method has been able to achieve the best solution and 0, otherwise.

Although ARPD and ARPT measures are commonly used for heuristics and metaheuristics, we applied this analysis for MILP and B&B because during the computational experiments, there have been several indications that BDP outperformed the results obtained by the other methods. Hence, we in order not to obtain negative percentages for ARPD, we compared the makespan obtained of a given method with the best makespan displayed out of all the tested methods.

After performing these calculations, the next step consists in verifying which test is appropriate for the analysis. The objective of this analysis is to determine if statistical differences between a certain indicator can be perceived when comparing different methods. For this research, Analysis of Variance (ANOVA) and Kruskal-Wallis test have been considered. The first one demands some conditions such as normality condition for the data, homoscedasticity and independent samples in order to be applied. In case one of these conditions fails, the analysis will rely on the Kruskal-Wallis test, which is non-parametric. In addition, for the latter it is also relevant to show a pair-wise comparison test in order to show which methods can present a statistical difference. The design of these tests are based on two hypotheses, which are commonly known as H_0 (null hypothesis) and H_1. The null hypothesis often states that whatever the factor used for comparison is, there exists no statistical difference between the samples and the contrary is stated by H_1. In order to verify these hypotheses, the p-value provided by the chosen test is provided and in case p-value is less or equal a significance level, denoted by α, the null hypothesis is rejected and the factors differ, otherwise the H_1 hypothesis is rejected and the factors do not present differences statistically. For the context of this research, the null hypothesis for Kruskal-Wallis test states that when comparing algorithms in a pairwise manner, they present no statistical difference.

To compose the statistical analysis, other features will also be considered by displaying graphic behaviour of the $ARPD$ measure varying according to the number of jobs, number of machines, ARPT and setup times configurations. Moreover, in order to enrich the study, a boxplot investigation will be presented so as to fully comprehend the ARPD's variability through the perspective of the algorithms designed in this research.

4. Computational Results

The content of this part is divided in two subsections that are relative to the experimental sets previously outlined. First, a discussion is carried out considering the analysis of the set containing 540 problems and afterwards a similar investigation is executed for the set with 2,880 problems.

4.1 First experimental set

For these experiments, data from 15 methods have been provided: MILP, B&B, and 13 variants of BDP considering window widths H varying from 1 to 20,000 and due to the authors' signature the BDP methods are denoted by BDP-SN(H) for a general notation. Additionally, the data that are handled for this first analysis are kept as a single group regardless the number of jobs, machines, setup configuration and similar attributes, being only categorized according to a given method. Table 1 shows the results of the first experimental set.

Table 1
General data for the first experimental set.

Method	SR (%)	ARPD (%)	ARPT
BDP-SN(1)	**0.740741**	**10.749**	**-0.99998**
BDP-SN(10)	6.111111	3.58	-0.99983
BDP-SN(50)	16.85185	1.6083	-0.99934
BDP-SN(100)	23.7037	1.1139	-0.99876
BDP-SN(250)	36.85185	0.7033	-0.99712
BDP-SN(500)	43.88889	0.5218	-0,99473
BDP-SN(750)	50.18519	0.4062	-0.9924
BDP-SN(1000)	53.33333	0.3542	-0.99004
BDP-SN(1250)	55.74074	0.3099	-0.98764
BDP-SN(2500)	65	0.2231	-0.97267
BDP-SN(5000)	72.59259	0.1514	-0.92404
BDP-SN(10000)	80	0.1136	-0.766
BDP-SN(20000)	**87.03704**	**0.0822**	-0.17259
B&B	**58.14815**	**1.1434**	**5.121619**
MILP	**46.2963**	**1.174**	**6.673523**

As it can be seen in Table 1, the descriptive analysis shows that as the success rate of BDP-SN (H) improves, the values of ARPD decrease and the ones relative to ARPT tend to increase. This statement is obvious in the sense that as window widths become larger, more vertices are allowed in the state space. Therefore, this increases the probability of finding the graph that contains the shortest path and, consequently, the optimal solution. However, this process also generates a computational burden, mostly for sorting and dominance conditions, causing an expansion regarding the ARPT. Thus, the best results in terms of SR and ARPD are provided by BDP-SN(20000), whereas its ARPT can be considered the worst among the BDP-SN(H). The opposite can also be stated for BDP-SN(1). Furthermore, the values relative to the B&B and the MILP are highlighted. The descriptive comparison shows that these methods may present some statistical difference regarding their solution quality to some of the bounded dynamic programming variations and clearly displayed far higher CPU times, which indicates that some of the BDP algorithms may produce a better set of solutions with less computational effort. In fact, if one were to compare, for instance, BDP-SN(2500),the success rate is superior when tested against both methods and so are the ARPD and ARPT. Additionally, when recording the CPU times of individual samples, for example, the largest portion of complex instances, such as $n = 20$ and $m = 4$, would reach optimum in about 40s while the MILP and B&B would reach time limit without obtaining the best makespan.

Although these results seem promising, for sake of completeness regarding the statements previously made, some analysis might be needed to corroborate them. The data of individual samples have been subjected to Anderson-Darling normality test and Bonferroni homoscedasticity test so as to decide whether ANOVA would be an appropriate test when comparing different methods. The first test has a null hypothesis for normality of data and the second one constructs a null hypothesis for equal variances. According to the results drawn from the statistical analysis, one can notice the p-values for the first test have confirmed that the data is not normally distributed (except for BDP-SN(1), whose p-value is 0.196), once those are less than 0.05 (significance level). Furthermore, the same pattern can be observed for the second test, indicating the conditions for the ANOVA does not hold and therefore, Kruskal-Wallis test is selected to carry out the analysis.

According to Kruskal-Wallis test performed for RPD, one can conclude that at least one of the medians differ significantly, since the overall p-value for groups is less than 0.05. In this case a *post hoc* test, commonly known as Dunn's Test is applied so as to establish pairwise comparisons between groups and define those that, in fact, differ statistically. It is also valid to highlight that, due to corrections inserted in the Dunn's test formulation, the value of the significance level α might be altered to fulfill the test's purpose. For this data set, Minitab yields $\alpha = 0.002$.

Out of the 105 possible combinations, 89 presented significant difference according to Dunn's test. BDP-SN(1) and BDP-SN(10) show these differences when tested against the other methods and this fact might be expected since their ARPD values are more scattered throughout the range of data and the medians drastically differ from the others. Furthermore, comparisons developed with the MILP method statistically differ from almost every BDP-SN algorithm and a similar behavior can be observed for the B&B. Regarding the ARPD measure, both methods are positioned between the BDP-SN algorithms, which means that there might be methods with better and worse ARPD. Methods with worse ARPD tend to exhibit data with higher variability, while those with better ARPD show reduced variability and, depending on how scattered these data are, even with the equal medians, the test acknowledges the existing difference between the paired methods. These claims are also endorsed by the boxplot analysis in Fig. 5.

The boxplot is shown in order to present statements about variability and disposition of data that concerns the RPD. It can be seen that variability reduces as window widths' size increases, once the makespan values of each sample are either

improved or kept at the same level of the previous window width. Furthermore, it is perceived that when a comparison with MILP is furnished, BDP-SN(50) produces a RPD range that is significantly shorter, despite a much lower success rate, indicating that even with a narrow window width, BDP approach is able to provide smaller gaps based on the best solution among all methods. Similarly, the comparison between medium sized window width BDP and B&B shows that the former presents data range with overall smaller gaps. It can also be recognized that the concentration of outliers seems more prominent for wider window widths, and this can be explained by the reduction in the boxplot for these methods. Once the scheduler allows larger window widths, the number of optimal solutions in the data rises, causing more RPD data to be equal zero and it alters the calculations for the quartile interval, which sets the limits for the boxes and the outliers. When this interval reaches zero, which can be observed for BDP-SN(10000) and BDP-SN(20000), the optimal makespan is ensured for at least 75% of the data associated to a given method.

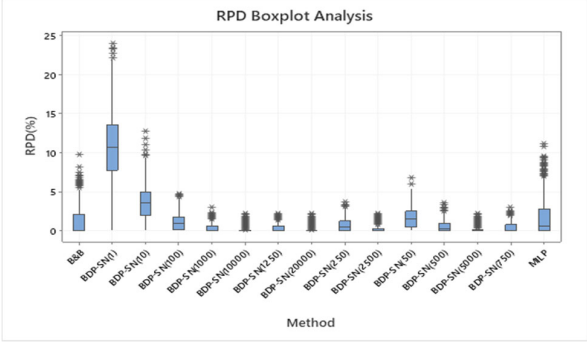

Fig. 5. Boxplot relative to the RPD data

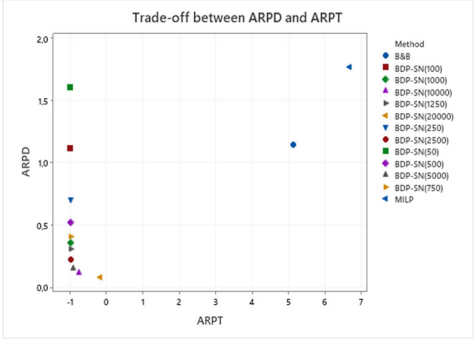

Fig. 6. Performance analysis with regard to ARPD and ARPT

The analysis of solution quality and average CPU time is imperative for this experimental set once that it helps the scheduler in the decision making process to optimize its policies based on a particular reasonable time frame. According to the aforementioned remarks, BDP-SN(1) and BDP-SN(10) will be neglected for this analysis, once they show discrepancies when contrasted with their counterparts. Results are depicted in Fig. 6 with the best trade-off being defined by the point closest to the leftmost bottom corner of the scatter plot, which is represented by BDP-SN(5000). According to the graphic description, there are points relatively close to BDP-SN(5000) that could also be adequate options to provide a reasonably good trade-off (e.g. BDP-SN(2500) and BDP-SN(10000)) however, the farther the points are from the best trade-off in terms of ARPD, the worse the success rate of a certain method. Analogously, the farther the points are in terms of ARPT, the higher the computational effort. In addition, one can tell that MILP and B&B are among the worst performances, since their location in the plot are not anywhere near those provided by BDP and despite their capacity to find optimal solutions, their average CPU times are significantly higher in comparison to some of the BDP approaches.

Fig. 7. ARPD variation with respect to the number of jobs

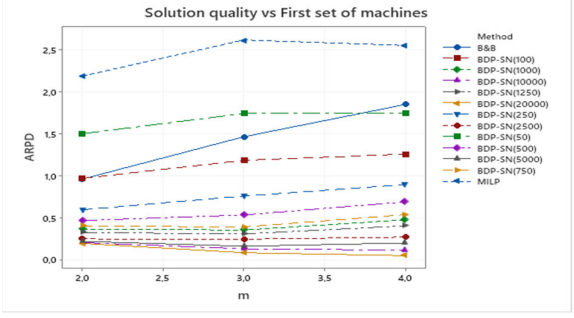

Fig. 8. ARPD variation with respect to the first group of machines

In order to complete the investigation for this model, it is also interesting to cover the behavior of solution quality considering the number of jobs, depicted in Fig. 7. It can be seen that this partitioned analysis follows the behavior of the general considerations outlined thus far. The Initially, MILP and B&B can be accounted for the best values for ARPD but for $n > 12$, an inversion occurs as ARPD for MILP and B&B increase at a rather elevated rate compared to the other algorithms. Furthermore, BDP-SN algorithms ranging from $H = 50$ to $H = 1250$ also show an increase in their growth rates, however they are far subtler than those obtained by B&B and MILP. Moreover, a decreasing rate can be observed in the BDP-SN(5000), BDP-SN(10000) and BDP-SN(20000) when $n > 16$, which may indicate that those are also more adequate for a larger set of jobs.

A similar analysis can be addressed in terms of number of machines but rather than investigating a single set of machines, the data was split in two parts because the first group is composed of 120 samples for each machine while the second presents 60 samples. In Fig. 8, it is observed MILP is the method with the highest variation in this group, although a slight

reduction is seen from $m = 3$ to $m = 4$. Also, B&B rapidly increases its ARPD value as the number of machines is expanded. The BDP-SN algorithms tend to maintain the behavior of differing slightly from one another in an increasing rate of ARPD for the small and medium sized window widths and a decreasing rate for larger window widths. The second group of machines is depicted in Fig. 9 and one can notice that for this set MILP and B&B perform better in terms of solution quality, however still varying at higher values than the majority of BDP-SN and once more, the BDP-SN with large window widths presents a slight decrease in the ARPD as the number of machines increases.

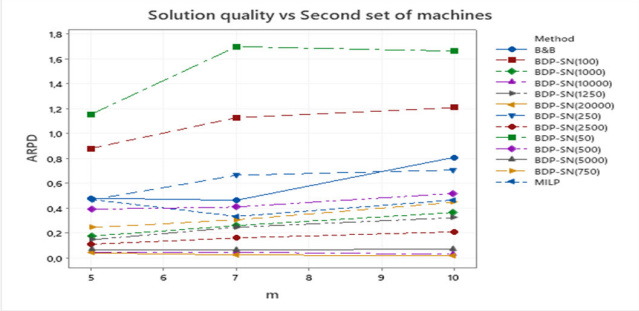

Fig. 9. ARPD variation with respect to the second group of machines

4.2 Second experimental set

This second set contains data from 13 methods, which can be divided into 12 variants of the BDP-SN(H) and a MILP procedure. For the former, only window widths that range from $H = 1$ to $H = 10000$ have been considered. BDP-SN(20000) has been excluded from the analysis due to the fact that an expressive amount of samples in the last group of jobs extrapolated the time limit established for the experiment.

Table 2
General data for the first experimental set.

Method	SR (%)	ARPD (%)	ARPT
BDP-SN(1)	**8.715278**	**8.21940**	**-0.9929**
BDP-SN(10)	37.84722	2.26220	-0.9767
BDP-SN(50)	54.61806	1.01650	-0.9601
BDP-SN(100)	60.34722	0.71440	-0.952298
BDP-SN(250)	66.07639	0.47730	-0.938084
BDP-SN(500)	70.9375	0.33860	-0.942152
BDP-SN(750)	73.88889	0.28010	-0.936395
BDP-SN(1000)	75.38194	0.24010	-0.933759
BDP-SN(1250)	76.28472	0.21590	-0.930479
BDP-SN(2500)	80.90278	0.14473	-0.910878
BDP-SN(5000)	86.04167	0.09276	-0.845928
BDP-SN(10000)	94.0625	0.05031	-0.643232
MILP	**78.64583**	**0.58730**	**10.9629**

The first analysis furnished for this set categorizes the general information according to the success rate, ARPD and ARPT of each method and it is summarized on Table 2. Note that, as expected, the larger the window width, the better the success rate related to an extension of the BDP algorithm. Additionally, one can notice that, even though the success rates presented by BDP-SN(1) and BDP-SN(10) are inferior compared to the other ones, a remarkable improvement can be observed when comparing to the first experimental set and this might indicate that the algorithm is susceptible to fast refinement of solutions, once the difference between their ARPTs is fairly small. It is also noticeable that solutions are upgraded for the medium-sized window widths, however it occurs at a smaller step between two consecutive BDP-SN methods. Regarding the MILP method, its results have produced a far superior success rate as well as ARPD than those presented by the first set of experiments, which is also seen in the other methods when compared to their counterparts in the first experimental set. This may also be explained due to the fact that the number of samples have grown considerably and therefore, so has the likelihood of problems that could achieve the optimal makespan. Furthermore, some intermediate BDP-SN algorithms such as BDP-SN(1000) and BDP-SN(1250), as it shall be shown later, might be comparable to the MILP performance, once a trade-off between solution quality and CPU time is developed. BDP-SN(2500), BDP-SN(5000) and BDP-SN(10000) have presented the best performances in terms of success rate and consequently, ARPD. It is also valid to point out that these methods have also been able to yield these results under a much smaller CPU time than the one imposed as a stopping criterion for the algorithms, while MILP has reached time limit for a variety of problems and has not been able to obtain the optimum for 615 of them.

Once again, it is paramount to show which algorithms differ statistically and prior to that, determine whether an ANOVA is adequate to define a comparison among the algorithms. In this case, the statistical analysis shows that the data does not follow normal distribution (p-value < 0.05 for Anderson-Darling Test) and the homoscedasticity cannot be confirmed (p-value < 0.05 for Bonferroni Test) and therefore, ANOVA is not compatible with the analysis. Thus, a Kruskal-Wallis test is performed with a *post hoc* investigation in order to establish an appropriate discussion and, as it has been previously stated for the first experimental set, the test is composed of a series of pairwise comparisons that rely on a different value of α, due to some corrections that need to be performed to validate the analysis. Hence, according to the calculations of Minitab, the value of α is set to 0.003 and therefore, the p-value is analyzed with respect to this value.

The pairwise comparison is composed of 70 samples, out of 78, that present statistical differences regarding the RPD measure. Although the comparison among the several BDP-SN methods is displayed, the interest here is to contrast them with MILP, once it has a different nature from the others. As expected, MILP can be distinguished from small and large-sized window widths BDP-SN methods. For the small ones, one can state that the high variability generates values for the median that are greater than zero and RPD data is more scattered than those belonging to MILP. For the large window widths, the median is indeed equal to that yielded by MILP, however the data is much more compressed and this factor is also accounted by for the test, hence demonstrating that a statistical difference occurs. Conversely, no difference can be accounted for when referring to MILP and medium-sized window widths BDP-SN algorithms and this could be explained by the fact that the slight rate of change between two consecutive BDP-SN methods may not be perceived drastically by Dunn's Test. This subtle change is actually an indication that only few improvements for makespan have taken place when switching to the next window width (for some of the medium-sized window widths) and these improvements might be closer to that obtained by MILP.

The analysis made via Kruskal-Wallis test can actually be corroborated by the Boxplot depicted in Fig. 10. It can be noticed that BDP-SN(1) and BDP-SN(10) are the algorithms that show the highest variability among all methods and consequently, deliver the worst results. This variability reduces drastically when $H = 50$ as a large portion of solutions are rapidly enhanced or reach the optimum. For $H \geq 1000$, the interquartile difference is zero, which is also the same for the MILP, however for the former one can notice that the number of outliers tend to diminish as H becomes larger and this phenomenon specifies the increase in the number of problems whose optimal solutions have been found. Furthermore, the outliers for the MILP method vary on a larger scale than those observed for BDP-SN whose $H \geq 1000$.

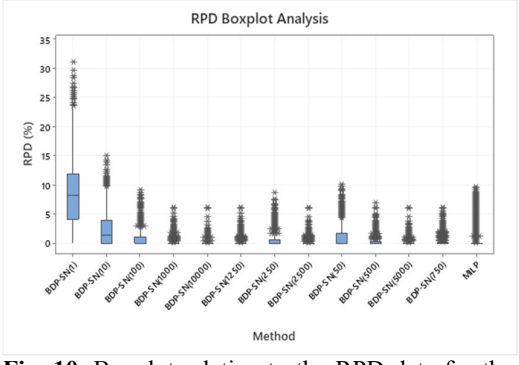

Fig. 10. Boxplot relative to the RPD data for the second experimental set

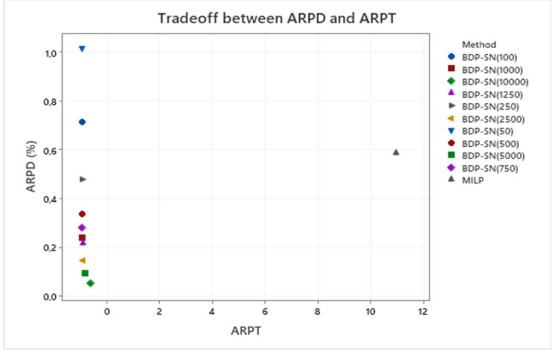

Fig. 11. Tradeoff with respect to ARPD and ARPT for the second experimental set

For this set of experiments, it is also relevant to establish a Pareto trade-off analysis in order to investigate which of the methods that have been developed for this problem may be the best choice for the scheduler. It is relevant to mention that the points relative to BDP-SN(1) and BDP-SN(10) have been neglected since their values show discrepancies when compared to the other algorithms. Fig. 11 represents the scatter plot adjusted according to the values of ARPD and ARPT seen in Table 2. The objective is to find the vector closest to $P(-1,0)$, once the values are asymptotic to $ARPD = 0$ and $ARPT = -1$. Therefore, by calculating the distance between a point relative to a given method and $P(-1,0)$, one can conclude that BDP-SN(5000) is the most efficient method among those that have been tested in this experimental set, since it is the closest to the reference point. Although MILP might not produce the farthest point, it can be observed that some BDP-SN algorithms (e.g. BDP-SN(1000), BDP-SN(1250), BDP-SN(2500), BDP-SN(10000)) are indeed better choices when compared to it because their ARPD is, in fact, smaller and the ARPT is much closer to the reference axis. In addition, some of these methods are comparable with MILP's performance, once they produce a much smaller ARPD whereas their success rate is gauged to be worse, yet relatively close.

The last analysis comprises the evaluation of the ARPD with respect to the variation in the number of jobs, variation in the number of machines and variation in the setup rates. The first situation is depicted in Fig. 12, and one can notice that for $n = 4$ and $n = 5$, every algorithm is able to maintain their ARPD levels. The MILP method is able to keep its ARPD at zero up to $n = 10$, which means that it responds well to variations in a small-sized job set and the same can be stated about the BDP-SN algorithms with large window widths. However, this scenario tends to change once a medium-sized job set is

analyzed and MILP presents a strong leap when varying jobs in this class of jobs while large window width BDP-SN algorithms tend to vary subtly when compared to the previous n analyzed. Note that for BDP-SN(10000), the ARPD values tend to reduce as the number of jobs increases. It is worth mentioning that, despite presenting continuous lines in the graph, there exists no guarantee that the behavior displayed by those lines is in fact the behavior of the system for jobs in-between and hence, the reader should only draw conclusions from the markers relative to the specific job set that has been used in the computational experiments.

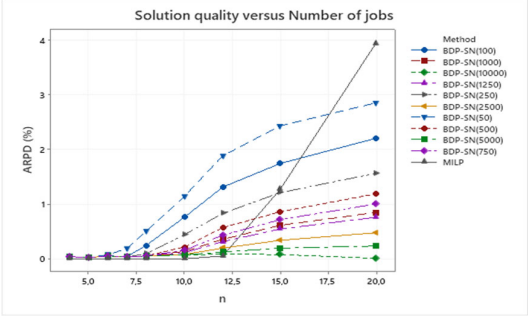

Fig. 12. ARPD variation with respect to the number of jobs for the second experimental set

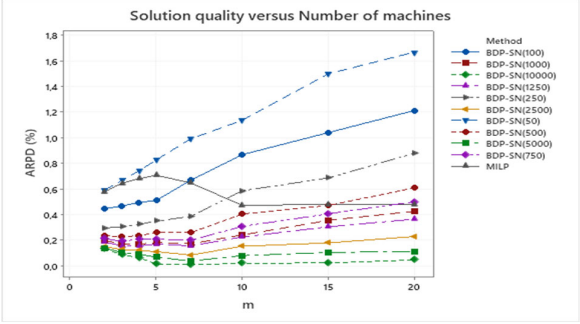

Fig. 13. ARPD variation with respect to the number of machines for the second experimental set

Fig. 13 depicts the behavior of ARPD regarding the variation in the number of machines. It is noticeable that for small window widths up to the ARPD tends to increase as number of machines in the given set becomes larger. Indeed, when undergoing the computational experiments, the BDP-SN algorithms that comprised these window widths did not present significant changes for this set of machines. For the MILP method, it has been shown that it tends to increase for the small-sized portion of the machine set, however for it responds quite well, even if slight changes are bound to occur for . For large window widths, one can observe that the algorithms tend to show a decreasing rate for the small-sized machines while they tend to increase for the medium-sized machines. Nevertheless, these increasing rates are not severe, which leads to satisfactory values of ARPD and consequently, to an excellent performance of these methods when compared to MILP and their smaller window width counterparts. In Fig. 14 it is shown how ARPD responds to the variation in setup rates. It is observed that the behavior is clearly the same for the BDP-SN algorithms, regardless of the window width and this is an expected outcome, once the nature of it is the same for the BDP-SN. Furthermore, the range of the ARPD is not drastically affected by altering the setup rates and it is noticed that BDP-SN tends to present better solutions for , once the ARPDs reach the minimum with regard to their respective window width. For MILP, the variation has a different effect, since as the setup rates grow larger, the values of ARPD increase rapidly and the changes can be clearly perceived.

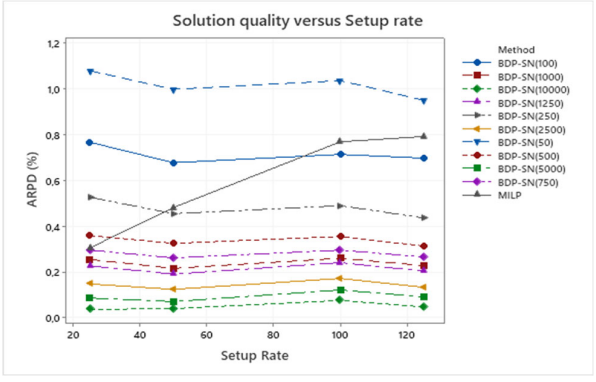

Fig. 14. ARPD variation with respect to the setup rates for the second experimental set

A final remark that needs to be done is a comparison between the two experimental sets that have been outlined in this subsection. When analyzing the graphs, one may realize that the plots with the same characteristics may differ, and it is relevant to mention that this is bound to happen because there are some factors that have influence on the results obtained experimentally. The first one is the number of samples for each algorithm. which are greater for the second set. Furthermore, when analyzing the solution quality regarding the number of machines there exists a rupture on the first experimental set due to the number of samples used for a given set. The number of machines is also another determinant factor to establish a difference between the two sets because the first one is bounded by at most 10 machines and the second comprises 20 machines at most. Finally, the first set is restricted to a single type of setup rates and the second set presents a variation of those and as it was previously shown, the alteration in these rates have an effect upon the solution quality.

5. Concluding Remarks

Scheduling problems have been one of the pillars with regard to general problems in Operations Research. Since it is a combinatorial problem, it is expected that any slight variation might affect majorly the complexity associated to a given ramification of a classic approach. Despite this fact, some trends in terms of objective functions and technological constraints have been placed in more contemporary problems integrated with other portions of the supply chain over the last ten years and consequently, the classic problems in scheduling have received little attention from the researchers.

Therefore, this research focuses on working on a classic problem that still plays a relevant role in the industrial scenario, which is the blocking flow-shop problem with sequence dependent setup times. Due to little occurrences in literature for this problem and therefore, scarce solution methods to deal with it, we decided to investigate some alternative options that could be fitting for the problem of interest. By researching some methods, it was noticeable that dynamic programming has been resurfacing and being aggregated to other algorithms in order to produce more efficient ones. Hence, after carrying out a thorough research, Bounded Dynamic Programming has been selected due to its results regarding blocking flow-shop and in more complex systems such as job-shops and open shops with the inclusion of sequence dependent setup times.

The mathematical analysis has been outlined by using the main concepts and propositions introduced by Joaquín Bautista et al. (2012) and a lower bound proposed by Takano and Nagano (2017) and the algorithmic structure has been delineated according to the junction of both sources. Moreover, some adjustments in the initial algorithm, which is similar to the one in Joaquín Bautista et al. (2012), so as to increase the algorithm's performance, once the setup inclusion generates a greater computational burden. These modifications are simply the ordering and the disregard of some functions that have been considered by the original authors and are related with sorting and state space reduction.

The experiments are divided into two sets. The first one is composed of 540 problems, whose division takes place according to $n = \{10,12,14,16,18,20\}$ and $m = \{2,3,4,5,7,10\}$. The BDP-SN, which is the method based on Bounded Dynamic Programming, is basically divided into 13 replicas, each containing a specific window width, and a comparison is established among them, a B&B algorithm and a MILP. The second set is defined analogously but each method is composed of 2880 problems and the comparison is established only with MILP.

The statistics involved in the experiments show that for the first set, the success rate can reach 87%, approximately and for the scenario involved, some of the BDP-SN algorithms would outperform the MILP and B&B in terms of success rate, ARPD and ARPT measures. Also, statistical difference, which is expected to happen, is confirmed for most methods and this is an initial indication of the best performance achieved by some of the BDP-SN methods when compared to the others. In addition, the boxplot analysis shows the evolution of the method by increasing the window widths and how the makespan is, in fact, improved and how the variability tends to diminish with the enlargement of window widths. In addition, the trade-off analysis evidences that BDP-SN(5000) is the best choice of algorithm among the ones proposed since the ARPD and ARPT present the shortest distance from the reference point, meaning that it can yield a small variation regarding the optimal solutions within a small amount of CPU time.

The statistical analysis is consonant with the one for the first set regarding the advantage of using some of the BDP-SN methods. Since the samples are larger, it is expected that the success rates increase and better values of ARPD and ARPT are obtained when comparing the first and second sets. The MILP presents quite good performance, being able to reach optimality for 78.65% of the samples, even though the average time associated with it is much larger when compared to the BDP-SN methods. The best one regarding BDP-SN is BDP-SN(10000), which is able to accurately reach the optimal solution for 2709 samples out of 2880 (approximately 94%). The Kruskal-Wallis analysis shows that 70 pairwise comparisons present statistical differences, which means that some BDP-SN might perform better than the MILP or even be a reasonable substitute in a trade-off analysis. Therefore, these assumptions are both confirmed in the boxplot evaluation and the trade-off analysis. The first one shows that good results in terms of variability can be obtained by using algorithms with window widths larger than 1000, once they have the same interquartile difference of MILP and their variability is way more reduced. Furthermore, the trade-off assessment shows that BDP-SN(5000) is the one with the best performance when analyzing both ARPD and ARPT.

No further analysis has been made, however some considerations must be made, once the method worked efficiently for the samples evaluated here. The first one is to expand the number of jobs and verify the strength of the algorithm for larger instances. Taking into account the work done by Joaquín Bautista et al. (2012), the BDP for the blocking flow shop has been applied to all the Taillard instances and this might be an indication that the BDP-SN can also obtain fairly good solutions for larger instances. Secondly, the lower bound is also another pathway that must be considered, since the BDP-SN strength can be measured by the tightness of the bound and further research might be developed in constructing a more efficient bound that can show the best trade-off between tightness and exactness. The last consideration is with regard to the upper bound, which here has relied on $H = 1$ for the BDP, however the development of more appropriate heuristics might increase the efficiency of the algorithm and enhance the number of optimal solutions for our benchmark and ones with larger number of jobs.

Since BDP-SN has yielded high quality solutions for the $Fm|s_{ijk}, block|C_{\max}$ it might be also suitable for other regular objective functions such as total tardiness, number of tardy jobs and total flow time. Additionally, once those have been proven efficient, BDP could be extended to more complex objective functions that are either multi-criteria or non-regular. In terms of refinements regarding the BDP structure, we have seen that BDP is dependent on efficient dominance rules so as to reduce the state space and therefore, research on developing new dominance rules might be useful in the algorithm's performance. Additionally, the window width has been defined according to previous sources and rather than setting it from static values, devising a dynamic procedure that is able to widen or shrink its size might help improve the solutions as well as the computational effort.

Acknowledgment

This research was supported by Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) – Brazil, under grant number 88882.379101/2019-01 and Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) - Brazil under grant number 312585/2021-7.

References

Agnetis, A. & Mosheiov, G. (2017). Scheduling with Job-Rejection and Position-Dependent Processing Times on Proportionate Flowshops. *Optimization Letters, 11*(4), 885–92.

Allahverdi, A. (2015). The Third Comprehensive Survey on Scheduling Problems with Setup Times/Costs. *European Journal of Operational Research, 246*(2), 345–78.

Allahverdi, A., Ng, C. T., Cheng, T. C. E. & Kovalyov, M. Y. (2008). A Survey of Scheduling Problems with Setup Times or Costs. *European Journal of Operational Research, 187*(3), 985–1032.

Bautista, J. & Pereira J. (2009). A Dynamic Programming Based Heuristic for the Assembly Line Balancing Problem. *European Journal of Operational Research, 194*(3), 787–94.

Bautista, J., Cano, A., Companys, R. & Ribas, I. (2012). Solving the Fm/ Block/ Cmax Problem Using Bounded Dynamic Programming. *Engineering Applications of Artificial Intelligence, 25*(6), 1235–45.

Fernandez-Viagas, V., Leisten, R. & Framinan, J. M. (2016). A Computational Evaluation of Constructive and Improvement Heuristics for the Blocking Flow Shop to Minimise Total Flowtime. *Expert Systems with Applications, 61*, 290–301.

Gong, H., Tang, L. & Duin, C. W. (2010). A Two-Stage Flow Shop Scheduling Problem on a Batching Machine and a Discrete Machine with Blocking and Shared Setup Times. *Computers & Operations Research, 37*(5), 960–69.

Graham, R. L., Lawler, E. L., Lenstra, J. K. & Kan, A. H. G. R. (1979). Optimization and Approximation in Deterministic Sequencing and Scheduling: A Survey. *Annals of Discrete Mathematics, 5*, 287–326.

Gupta, J. N. D. & Stafford Jr., E. F. (2006). Flowshop Scheduling Research After Five Decades. *European Journal of Operational Research, 169*(3), 699–711.

Hon, K. K. B. (2005). Performance and Evaluation of Manufacturing Systems. *CIRP Annals, 54*(2), 139–54.

Ignall, E. & Schrage, L. (1965). Application of the Branch and Bound Technique to Some Flow-Shop Scheduling Problems. *Operations Research, 13*(3), 400–412.

Johnson, S. M. (1954). Optimal Two-and Three-Stage Production Schedules with Setup Times Included. *Naval Research Logistics Quarterly, 1*(1), 61–68.

Koren, Y., Heisel, U., Jovane, F., Moriwaki, T., Pritschow, G., Ulsoy, G. & Brussel H. V. (1999). Reconfigurable Manufacturing Systems. *CIRP Annals, 48*(2), 527–40.

Lawler, E. L. & Moore, J. M. (1969). A Functional Equation and Its Application to Resource Allocation and Sequencing Problems. *Management Science, 16*(1), 77–84.

Lepuschitz, W., Zoitl, A., Vallée, M. & Merdan, M. (2010). Toward Self-Reconfiguration of Manufacturing Systems Using Automation Agents. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews), 41*(1), 52–69.

Li, S. S. & Yuan, J. J. (2020). Single-Machine Scheduling with Multi-Agents to Minimize Total Weighted Late Work. *Journal of Scheduling, 23*, 497–512.

Maccarthy, B. L. & Liu, J. (1993). Addressing the Gap in Scheduling Research: A Review of Optimization and Heuristic Methods in Production Scheduling. *The International Journal of Production Research, 31*(1), 59–79.

Mahalik, N. P. & Nambiar, A. N. (2010). Trends in Food Packaging and Manufacturing Systems and Technology. *Trends in Food Science & Technology, 21*(3), 117–28.

Martinez, S., Dauzère-Pérès, S., Gueret, C., Mati, Y. & Sauer., N. (2006). Complexity of Flowshop Scheduling Problems with a New Blocking Constraint. *European Journal of Operational Research, 169*(3), 855–64.

Merchan, A. F. & Maravelias, C. T. (2016). Preprocessing and Tightening Methods for Time-Indexed MIP Chemical Production Scheduling Models. *Computers & Chemical Engineering, 84*, 516–35.

Miyata, H. H. & Nagano, M. S. (2019). The Blocking Flow Shop Scheduling Problem: A Comprehensive and Conceptual Review. *Expert Systems with Applications, 137*, 130–56.

Moore, J. M. (1968). An n Job, One Machine Sequencing Algorithm for Minimizing the Number of Late Jobs. *Management Science, 15*(1), 102–9.

Mor, B. & Shapira., D. (2020). Regular Scheduling Measures on Proportionate Flowshop with Job Rejection. *Computational and Applied Mathematics, 39*(2), 1–14.

Newton, M. A. H., Riahi, V., Su, K. & Sattar, A. (2019). Scheduling Blocking Flowshops with Setup Times via Constraint Guided and Accelerated Local Search. *Computers & Operations Research, 109*, 64–76.

Ozolins, A. (2018). Bounded Dynamic Programming Algorithm for the Job Shop Problem with Sequence Dependent Setup Times. *Operational Research, 20*, 1701–1728.

Ozolins, A. (2019a). Dynamic Programming Approach for Solving the Open Shop Problem. *Central European Journal of Operations Research, 29*, 291–306.

Ozolins, A. (2019b). Improved Bounded Dynamic Programming Algorithm for Solving the Blocking Flow Shop Problem. *Central European Journal of Operations Research, 27*(1), 15–38.

Pinedo, M. (2012). *Scheduling*. Springer.

Potts, C. N. & Wassenhove, L. N. V. (1985). A Branch and Bound Algorithm for the Total Weighted Tardiness Problem. *Operations Research, 33*(2), 363–77.

Ribas, I., Companys, R. & Tort-Martorell, X. (2015). An Efficient Discrete Artificial Bee Colony Algorithm for the Blocking Flow Shop Problem with Total Flowtime Minimization. *Expert Systems with Applications, 42*(15-16), 6155–67.

Ronconi, D. P. (2005). A Branch-and-Bound Algorithm to Minimize the Makespan in a Flowshop with Blocking. *Annals of Operations Research, 138*(1), 53–65.

Seow, Y. & Rahimifard, S. (2011). A Framework for Modelling Energy Consumption Within Manufacturing Systems. *CIRP Journal of Manufacturing Science and Technology, 4*(3), 258–64.

Shao, Z., Pi, D. & Shao, W. (2018). A Novel Discrete Water Wave Optimization Algorithm for Blocking Flow-Shop Scheduling Problem with Sequence-Dependent Setup Times. *Swarm and Evolutionary Computation, 40*, 53–75.

Süer, G. A., Báez, E. & Czajkiewicz, Z. (1993). Minimizing the Number of Tardy Jobs in Identical Machine Scheduling. *Computers & Industrial Engineering, 25*(1-4), 243–46.

Takano, M. I. & Nagano, M. S. (2017). A Branch-and-Bound Method to Minimize the Makespan in a Permutation Flow Shop with Blocking and Setup Times. *Cogent Engineering, 4*(1), 1389638.

Takano, M. I. & Nagano, M. S. (2019). Evaluating the Performance of Constructive Heuristics for the Blocking Flow Shop Scheduling Problem with Setup Times. *International Journal of Industrial Engineering Computations, 10*(1), 37–50.

Tomazella, C. P. & Nagano, M. S. (2020). A Comprehensive Review of Branch-and-Bound Algorithms: Guidelines and Directions for Further Research on the Flowshop Scheduling Problem. *Expert Systems with Applications, 158*, 113556.

Wang, D.J., Yin, Y. & Liu, M. (2016). Bicriteria Scheduling Problems Involving Job Rejection, Controllable Processing Times and Rate-Modifying Activity. *International Journal of Production Research, 54*(12), 3691–3705.