Expert Systems With Applications 228 (2023) 120191

Contents lists available at ScienceDirect

Expert Systems With Applications

journal homepage: www.elsevier.com/locate/eswa

Solving the integrated planning and scheduling problem using variable neighborhood search based algorithms

Mário Manuel Silva Leite [a], Telmo Miguel Pires Pinto [b,a,*], Cláudio Manuel Martins Alves [a]

[a] ALGORITMI Research Centre/LASI, University of Minho, Campus de Gualtar, 4710-057 Braga, Portugal
[b] University of Coimbra, CEMMPRE, Paço das Escolas, 3004-531 Coimbra, Portugal

ARTICLE INFO

Keywords:
Production planning and scheduling
Variable neighborhood search
Identical parallel machines scheduling

ABSTRACT

In this paper, we address the Integrated Planning and Scheduling Problem (IPSP) on parallel and identical machines. Planning and scheduling are essential for the efficient management of supply chains. Although both pursue the same general objective, they are usually performed independently mostly because they relate to different timescales. As a consequence, the generated plans and schedules are typically sub-optimal from a global standpoint.

The approaches followed in this paper explicitly consider the interdependence between the planning and scheduling activities by solving them simultaneously in an integrated way. We explore different heuristics based on variable neighborhood search procedures with new and specifically designed neighborhood structures relying on the properties of the IPSP. The quality of these approaches is evaluated through extensive computational experiments performed on a large set of benchmark instances. The results show that the proposed methods achieve high-quality solutions, with a substantially low computation time, outperforming other state-of-the-art results reported in the literature.

1. Introduction

In the last years, several approaches have been developed to address the integrated optimization of two or more processes related to the management of supply chains. Common examples include the combination of production activities with distribution (Armstrong et al., 2008; Geismar et al., 2008), material procurement with the development of new products (Cheaitou & Khan, 2015), and planning with scheduling (Chen, 2016; Grossmann, 2009). In this paper, we address specifically this latter case. The production flows in a wide range of industrial settings depend heavily on the quality of the production plans and schedules. In most of the cases, these plans and schedules are built independently. Naturally, this approach leads to solutions which are sub-optimal most of the time (Shen et al., 2006). Since the plans developed in the planning phase are used as an input to build the production schedules subsequently, the only way to achieve solutions which are globally optimal is to explicitly consider this interdependence by solving the two related optimization problems simultaneously.

Planning and scheduling optimization problems arise in many real-world scenarios, such as in the make-and-pack dairy production of perishable food products. These products must be packed as soon as they are produced to avoid contamination. In Sel et al. (2017), the planning determines the amount of yogurt that should be produced during the mixing phase, while scheduling deals with the packaging part of the process by assigning the blend to the packaging parallel lines. The time spent between these phases must be reduced since the intermediate product has a short lifetime. For this purpose, planning and scheduling must be performed in tight connection to balance adequately the quantity of intermediate product (the product resulting from the mixture) with the availability of the packaging lines. Similarly, in a Batch-Processing Machines (BPM) problem, the interdependence of scheduling decisions is also verified. In the first phase, the batches are composed (*i.e.*, the jobs that will be part of each batch are identified) while ensuring the capacity of the machines. Then, the best job order is sought in the second phase to reduce the makespan (Zhou et al., 2021). Thus, it is clear that the possibilities for reducing the makespan (sorting decisions) are strongly conditioned by the previous decisions concerning the formation of batches.

In Dogan and Grossmann (2006) and in Erdirik-Dogan and Grossmann (2008), the authors address the simultaneous planning and scheduling of single-stage continuous multi-product plants in the chemical industry. The primary objective is to meet the due dates and to maximize the profit by considering different factors such as the constant production rates, the production costs, the selling prices, the inventory costs and the sequence-dependent transition times between products.

* Corresponding author at: University of Coimbra, CEMMPRE, Paço das Escolas, 3004-531 Coimbra, Portugal.
E-mail addresses: mario.leite@dps.uminho.pt (M.M.S. Leite), telmo.pinto@uc.pt (T.M.P. Pinto), claudio@dps.uminho.pt (C.M.M. Alves).

https://doi.org/10.1016/j.eswa.2023.120191
Received 13 July 2022; Received in revised form 25 March 2023; Accepted 15 April 2023
Available online 4 May 2023

In this case, planning consists in determining the products and the quantities that should be produced to satisfy both the demand and the inventory level in each week, while the objective of the scheduling problem is to determine the production sequence and the length of the production periods taking into account the transition costs.

A case from the glass industry was explored in Almada-Lobo et al. (2008). In the planning phase, colors and product lots are assigned to the furnaces, while scheduling deals with the programming of the color campaigns within each furnace. Planning is done for a full year time horizon. The integration of lot-sizing with scheduling is justified by the high sequence-dependent setup times in color changeovers. The iron and steel industry is also a no-wait scheduling process where the production has to deal with several operation modes and different power modes (temperature levels). In Zhao, He, and Wang (2021), the authors developed a two-stage cooperative evolutionary algorithm to tackle to the energy-efficient scheduling of the no-wait flow-shop problem. Concerns about energy saving are increasing, so the authors seek to minimize not only the makespan but also the total energy consumption.

Applications of planning and scheduling problems are not limited to the production field. They arise in different other contexts (Shen et al., 2006), as in hospitals, for example, to manage the execution of surgical interventions (Cardoen et al., 2010; Eun et al., 2019), or to allocate the nursing staff (Maenhout & Vanhoucke, 2013). We can also see it in applications of electric mobility, with the inherent concerns in charging battery vehicles. Liu et al. (2020) presents Electric Vehicle Charging Scheduling (EVCS) problem that considers the dependence among the station selection, charging options at each station, and the charging amount settings. In other cases, the production process is considered together with delivery. In Persson and Göthe-Lundgren (2005), the authors address the case of an oil refinery company through the joint analysis of production scheduling and shipment planning. The ships are used to bring the products from the oil refineries to the storage depots. Another example of combining production with distribution is described in Kopanos et al. (2012). In this case, a company that produces different types of dairy products has to schedule its production by taking into account the impact of the generated flows on the distribution operations downward. A critical aspect in this setting is the requirements of the quality of the product packaging.

In this paper, we address the Integrated Planning and Scheduling Problem (IPSP) on parallel and identical machines exactly as it was proposed first in Kis and Kovács (2012). The problem considers simultaneously the medium-term tactical planning of production operations with the short-term scheduling of production jobs, such that the output of the first is the input of the second (Maravelias & Sung, 2009). Due to the nature of planning and scheduling problems, they tend to be analyzed separately in the literature. The complexity of both problems can also explain this separation since they are NP-hard, which turns IPSP computationally intractable for large-size instances. However, this separation can fail when trying to find globally optimal solutions. Additionally, neglecting their integration can lead to negative effects: if a disruption conducts to changes in the allocation of resources, all planning must be revised. On the other hand, production planning disconnected from the schedule can lead to delays that could be avoided if there was a real integration (Kis & Kovács, 2012).

In the IPSP on parallel and identical machines, the jobs are processed on machines with limited capacity and identical. The time period in which a job is to be processed is determined in the planning phase. The scheduling part of the problem aims to assign the jobs to the available machines, and to sequence them. A job is characterized through its release date, due date, processing time and the penalties that apply whenever a job is processed before or after its due date. The jobs must be assigned to a machine and to a time period within the planning horizon such that the sum of all the penalties is minimized.

The solution approaches described in the literature for the IPSP are still rare (Kis & Kovács, 2012; Rietz et al., 2016). Kis and Kovács (2012) proposed an exact solution approach based on branch-and-cut, a monolithic integer program and two hierarchical decomposition approaches. The authors tested their approaches on an extensive set of random instances. The average computing time for their best approach remains higher than 540 seconds, with some executions reaching the imposed time limit of 20 minutes without achieving optimality.

An alternative exact solution approach for the IPSP was proposed in Rietz et al. (2016). The problem was formulated through a pseudo-polynomial network flow model, which is considered in its entirety in contrast with the model presented in Kis and Kovács (2012). Different strategies were explored to reduce the number of constraints and simplify the model. In a large number of cases, their approach was able to reach the optimal solution with an average execution time of approximately 6 minutes.

From the analysis of the state of the art, it becomes clear that despite the worthy contributions in the literature, solving the IPSP problem using exact methods proved to be challenging. Indeed, an exact method is able to solve small-size instances, but it is more time-consuming (or even computationally intractable) in larger instances. This reason strongly motivated our approach, aiming to achieve good-quality solutions in a reasonable time. This feature is suitable for problems requiring several re-optimizations or real-time optimization.

In this paper, we explore different Variable Neighborhood Search (VNS) based algorithms to solve the IPSP. In Section 2, the defining characteristics of the IPSP are presented and discussed. In Section 3, the different elements of our solution approach are presented, including the method used to obtain the initial solution, the neighborhood structures, and the variant of the VNS algorithm that we considered. In Section 4, preliminary tests were performed to adjust the problem parameters, and then we report on an extensive set of computational experiments conducted on benchmark instances. The results were discussed and assessed through statistical tools in the same Section. Finally, conclusions are drawn in Section 5.

2. Parallel machine integrated planning and scheduling problem

The Integrated Planning and Scheduling Problem (IPSP) described in Kis and Kovács (2012) consists in determining when and where a set of jobs should be processed so that the associated due dates are met as best as possible. Let T denote the time horizon within which the jobs must be processed. We divide the time horizon T into τ identical periods ($T = 1, \ldots, \tau$), each one with length $P \in \mathbb{N}$. The jobs are processed on machines which are assumed to be all identical. Let M be the set of machines. Each machine can only process one job at a time, and each job can only be processed once in a single machine and in a single time period. The machines operate in parallel mode. A job processing cannot be interrupted. The set of jobs will be denoted by N. A machine takes p_j units of time to process a given job $j \in N$. Clearly, a subset of jobs can be processed in a given period of time $t \in T$ if the total processing times of these jobs is not larger than P. Hence, the feasibility of a given instance of the IPSP requires that $P \geq max\{p_j : j \in N\}$. For each job $j \in N$, there is a release and a due date, which will be denoted respectively by $r_j \in T$ and $d_j \in T$. A release date r_j represents the earliest time period from which the job $j \in N$ becomes available for processing. Therefore, a job $j \in N$ can only be processed in a time period $t \in T$ if $t \geq r_j$. A due date $d_j \in T$ is the time period when the job $j \in N$ is expected to be completed. From this point forward, we will assume that $d_j \geq r_j$, $\forall j \in N$. Exceeding a due date or completing a job before its due date are both penalized through a given cost. The penalty that will apply whenever a job $j \in N$ is completed before its due date (earliness) will be denoted by $e_j \in \mathbb{N}$, while the penalty for lateness will be denoted by $l_j \in \mathbb{N}$. The final value of the penalty applied to a job $j \in N$ which is processed in a time period $t \in T$ will be given by:

$$w_t^j = e_j \times max\{0, d_j - t\} + l_j \times max\{0, t - d_j\}. \tag{1}$$

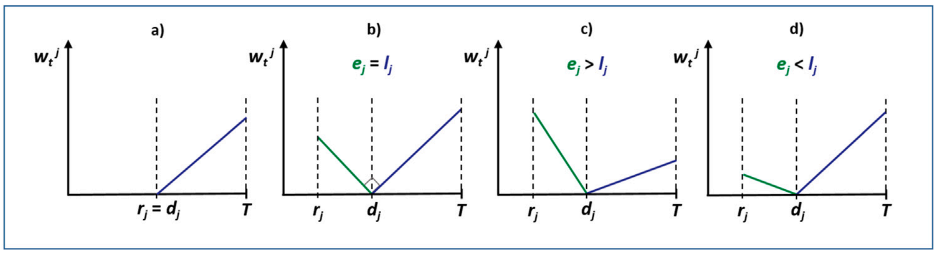


Fig. 1. Variation of the penalty for a job $j \in N$ along the planning horizon.

Table 1
Attributes for each job $j \in N$.

j	r_j	d_j	p_j	e_j	l_j
1	1	1	3	0	10
2	1	1	2	0	5
3	1	1	2	0	12
4	1	1	4	0	10
5	1	1	4	0	9
6	1	2	2	10	10
7	2	2	5	0	5
8	1	3	2	4	8
9	2	3	1	7	12
10	3	3	4	0	10
11	1	1	2	0	3
12	1	3	3	5	8

The objective of the IPSP is to determine the time period when each job will be processed, and the machine where each one will be completed, such that the sum of all the penalties (Eq. (2)) is minimized. The $p^i \in T$ refers to the time period in which job i is processed.

$$z(S) = \sum_{i=1}^{|N|} w^i_{p^i}. \tag{2}$$

For the sake of clarity, in Fig. 1, we represent the variations of the penalties along the time horizon in four different scenarios. If the release and due dates are the same, a penalty is applied only if the job is processed after that date (a). For some jobs, the earliness and tardiness penalty factors are the same, as in (b), while for other jobs the earliness and the tardiness may have a different impact on the penalty to be paid, as depicted in (c) and (d).

Example 2.1. Consider the case in which 12 jobs ($N = \{1, 2, \dots, 12\}$) must be processed in one of two parallel and identical machines ($M = \{1, 2\}$). The time horizon is set to one day (24 hours), and it is further divided into three 8-hours shifts. In this case, a shift corresponds to a time period and, hence, the time horizon of the problem is defined by $T = \{1, 2, 3\}$, with $P = 8$ and $\tau = 3$.

Table 1 lists all the parameters associated to the jobs $j \in N$. The processing times are given in hours, while both the due and release dates are expressed in time periods (shifts). The penalties for both earliness and lateness are also provided. In Fig. 2, a possible solution for this instance of the IPSP is depicted.

There is a penalty associated to the late processing of job 5, since it is performed one time period after its due date ($t = 2$), resulting in a penalty

$$w^5_2 = 0 \times max\{0, 1 - 2\} + 9 \times max\{0, 2 - 1\} = 9.$$

In contrast, job 8 is processed in a time period prior to its due date ($w^8_2 = 4$). The delay of two time periods for processing job 11 results in a penalty of 6 units. All the other jobs are processed in their due dates, and thus no additional penalties are applied. The sum of all penalties is equal to 19.

2.1. Integer programming model

This subsection presents the integer programming model proposed by Kis and Kovács (2012) to tackle the IPSP. The decision variables are x^j_{kt}, for $j \in N$, $k \in M$ and $t \in T$, taking value 1 if job j is assigned to machine k in period t, taking value 0 otherwise. Therefore, the IPSP can be formulated as follows:

$$\min \sum_{j=1}^{|N|} \sum_{k=1}^{|M|} \sum_{t=1}^{\tau} w^j_t x^j_{kt} \tag{3}$$

subject to:

$$\sum_{k=1}^{|M|} \sum_{t=1}^{\tau} x^j_{kt} = 1, \forall j \in N, \tag{4}$$

$$\sum_{j=1}^{|N|} p_{kt} x^j_{kt} \le P, \forall k \in M, t \in T, \tag{5}$$

$$x^j_{kt} = 0, \forall j \in N, k \in M, t < r_j, \tag{6}$$

$$x^j_{kt} \in \{0, 1\}, \forall j \in N, k \in M, t \in T. \tag{7}$$

The weights in the objective function are penalties incurred by each job according to Eq. (1) defined above. In a feasible solution, for each job j, the binary decision variable x^j_{kt} takes the value 1 for the one and only one combination of machine and time period, and it takes the value 0 for all other variables for the same job (constraints (4)). The capacity constraints for each machine and time period are ensured in inequalities (5). Constraints (6) ensure that all decision variables x^j_{kt} take value 0 in all periods t prior of the release date of the job j.

3. Variable neighborhood search approaches

As referred to in Section 1, the computing time required by the exact solution algorithms devised for the IPSP tends to increase significantly with the size of the instances. Even in the case of medium-size instances as those reported in Rietz et al. (2016), these approaches fail in many cases to find a proven optimal solution within reasonable time bounds. This observation motivates the exploration of heuristic procedures that might be able to efficiently solve larger instances by incorporating sophisticated search strategies.

Here, we explore in particular different local search procedures that we embed in alternative variants of the VNS algorithm. VNS starts its search with a given initial solution, and seeks for improvements iteratively through systematic moves within the spaces defined from the neighborhood structures. Local search is of paramount importance in this process as it may determine the capacity of the algorithm to escape from local optima. The VNS metaheuristic was first described in Mladenović and Hansen (1997) as an approach that favors the diversification of the search to avoid being stuck in a poor local optimum. Instead of stopping at local optima, VNS algorithms execute successive neighborhood exchanges that drive the search into a broader solution space, and thus increase their chances of finding better solutions. Cycles in the search process may be avoided through the combination of

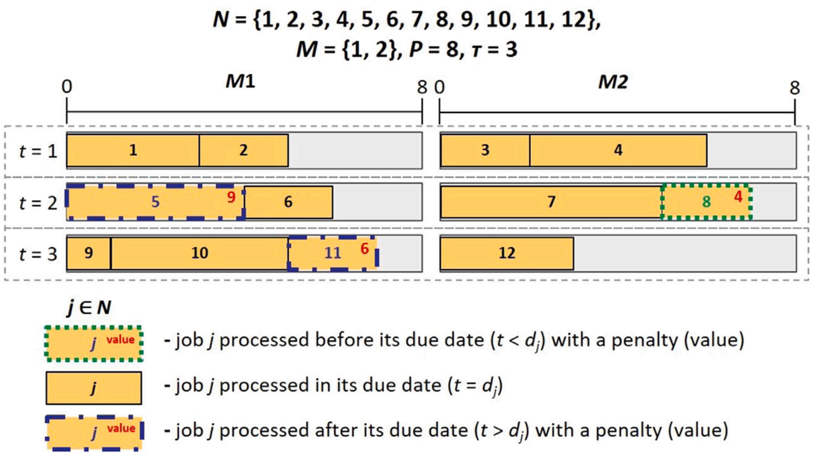

N = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12},
M = {1, 2}, P = 8, τ = 3
M1
M2
0
8
t = 1
t = 2
t = 3
1
2
3
4
5
9
6
7
8
4
9
10
11
6
12
j ∈ N
j value
- job j processed before its due date (t < dj) with a penalty (value)
j
- job j processed in its due date (t = dj)
j value
- job j processed after its due date (t > dj) with a penalty (value)

Fig. 2. Feasible solution of an instance of the IPSP (Example 2.1).

deterministic and stochastic neighborhood exchanges both in the local search and the shaking phase. Over the last years, several variants of the VNS metaheuristic have been proposed and applied to the most diverse set of problems (Almada-Lobo et al., 2008; Bilyk & Mönch, 2012; Brimberg et al., 2017; Hansen et al., 2017; Macedo et al., 2015; Zhao, Zhang, et al., 2021).

In the sequel, we present and describe all the elements that characterize the methods we devised to solve the IPSP. First, we introduce the approaches followed to represent the solutions by detailing the construction of the initial solutions and the neighborhood structures. Next, we discuss the variants of the VNS algorithm that we implemented to solve the problem under study. We explored several VNS approaches (Hansen et al., 2017) including both Basic and General Variable Neighborhood Search (BVNS and GVNS, respectively). Additionally, we developed and tested a skewed version of GVNS (Brimberg et al., 2017) which is also described below.

3.1. Representation of a solution and evaluation function

A feasible solution S of the IPSP can be represented by a set of $|N|$ elements such that

$$S = \{J^1, \dots, J^{|N|}\}.$$

For each job $i \in N$, let

$$J^i = (m^i, p^i, mp^i),$$

where $m^i \in M$ is the machine that processes job i, $p^i \in T$ is the time period in which job i is processed, and mp^i is the instant in period p^i when job i starts to be processed ($0 \leq mp^i < P$). Therefore, a solution of the IPSP can be represented as follows:

$$S = \{(m^1, p^1, mp^1), (m^2, p^2, mp^2), \dots, (m^{|N|}, p^{|N|}, mp^{|N|})\}.$$

As we will show later, we keep the time instant when the job starts to be processed, because this attribute is used to define one of our neighborhoods. Nevertheless, we must stress that the penalty (1) depends only on the time period in which the job is processed. The time instant when the processing starts has no impact on the objective function. The same happens with the processing order of jobs within the same time period: changing such a sequence has no impact on the penalties which are due.

The value of the objective function is the sum of all the penalties. Clearly, if a job is processed at its due date, no penalty applies. The expression of objective function is the same as defined in Eq. (2).

For instance, the solution of Example 2.1 depicted in Fig. 2 can be represented as follows:

$$\begin{aligned} S = \{&(1,1,0),\\ &(1,1,3),\\ &(2,1,0),\\ &(2,1,2),\\ &(1,2,0),\\ &(1,2,4),\\ &(2,2,0),\\ &(2,2,5),\\ &(1,3,0),\\ &(1,2,1),\\ &(1,3,5),\\ &(2,3,0)\}\end{aligned}$$

The value of the objective function in this case is given by:

$$z(S) = w_2^5 + w_2^8 + w_3^{11} = 19.$$

3.2. Computing an initial solution

To build an initial feasible solution for the IPSP, we resort to a greedy constructive heuristic which is summarized in Algorithm 1. First, the set N of jobs is sorted following a given criterion c previously defined (Line 2 – $SortList(N, c)$). Our criterion c is the decreasing order of the corresponding weighted penalty $(e_j + l_j)/p_j$, breaking ties by choosing first the job j with the largest processing time p_j. Following this order, each job j is assigned to a machine m in a period t that is equal to its due date (Line 9 – $AddJobToSolution(S, m, t, j)$), only if there is available capacity in that period without modifying any previous allocation. If a job cannot be assigned to the time period corresponding to its due date, it is discarded to be analyzed later in a second stage. Once all the jobs in N_{sorted} have been analyzed, those which failed to be allocated in the first stage are re-evaluated in the same order as defined initially (Line 15). If the due date of a selected job j is equal to its release date, the job is assigned to the earliest time period following its release (and due) date which has enough capacity. If the due date of the selected job j is different from its release date, then two penalties are computed: the penalty of assigning the job to the first available period prior to its due date, and another by assigning that job to the first available period after its due date. The job is finally allocated to the time period that yields the lowest penalty. The rationale behind this rule is simple, and consists in choosing the most favorable case between earliness and lateness, since they cannot be both avoided (Line 16-18).

```
Algorithm 1: Constructive Heuristic Algorithm
  Input:
    Instance for IPSP with a set of machines $M$, a set of periods $T$, and a set of jobs $N$;
    The sort criteria $c$;
  Output:
    A feasible solution $S$;
  Auxiliary:
    jobPlaced $\in \mathbb{N}$; m $\in M$; t $\in T$ and j $\in N$;

  Function ConstructiveHeuristic($M$, $T$, $N$, $c$)
1   S:=Ø;                                                   /* first stage */
2   $N_{sorted}$ := $SortList(N, c)$ ;
3   $N_{placed}$ := Ø;
4   while $j$ in $N_{sorted}$ do
5     $jobPlaced$ := 0;
6     while $m$ in $M$ and $jobPlaced = 0$ do
7       $t := GetJobDueDate(j)$;
8       if machine has available capacity in period t to process job j then
9         $S := AddJobToSolution(S, m, t, j)$;
10        $N_{placed} := AddJobToSet(N_{placed}, j)$;
11        $jobPlaced := 1$;
12      end
13    end
14  end
                                                            /* second stage */
15  while $j$ in $N_{sorted} \setminus N_{placed}$ do
16    $m := GetLowestPenaltyMachine(S, j)$;
17    $t := GetLowestPenaltyPeriod(S, j)$;
18    $S := AddJobToSolution(S, m, t, j)$;
19  end
20  return $S$
```

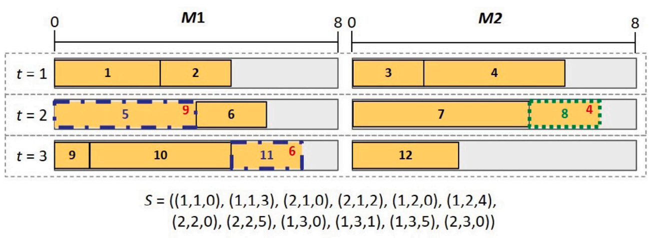

M1
M2
0
8
t = 1
t = 2
t = 3
1
2
3
4
5
9
6
7
8
4
9
10
11
6
12
S = ((1,1,0), (1,1,3), (2,1,0), (2,1,2), (1,2,0), (1,2,4), (2,2,0), (2,2,5), (1,3,0), (1,3,1), (1,3,5), (2,3,0))

Fig. 3. A feasible solution S.

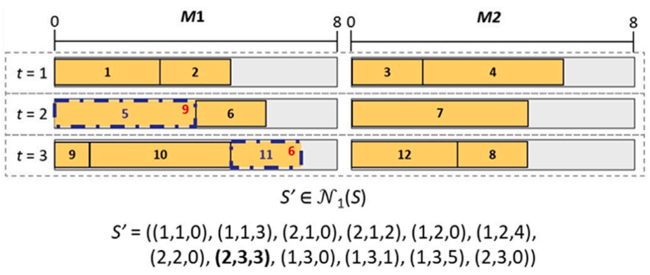

M1
M2
0
8
t = 1
t = 2
t = 3
1
2
3
4
5
9
6
7
9
10
11
6
12
8
S′ ∈ N1(S)
S′ = ((1,1,0), (1,1,3), (2,1,0), (2,1,2), (1,2,0), (1,2,4), (2,2,0), (2,3,3), (1,3,0), (1,3,1), (1,3,5), (2,3,0))

Fig. 4. Example of a neighbor $S' \in \mathcal{N}_1(S)$ of the solution S (as presented in Fig. 3).

3.3. Neighborhood structures

In our implementation, we considered six different neighborhood structures, which are described next. Some of them are used both at the shaking and local search phase of the VNS algorithms, while others are used exclusively at one of these phases. Only feasible solutions are explored. Note that the value of a neighbor solution can be only different from the value of a given current solution if at least one job is moved from its current period to another. For instance, moving one job to a different machine within the same period has no impact on the value of the solution. However, it is, in fact, a different solution, and thus it can be an output of the shaking phase.

To illustrate the presentation of the neighborhood structures, we will use the small instance of Example 2.1, and in particular its feasible solution S depicted in Fig. 3. Therefore, a neighbor solution (S') of the original solution (S) is presented for each neighborhood structure.

- $\mathcal{N}_1$: *Moving one job to a different time period*
  The neighbors of a solution S are obtained by moving one job from its current time period to a different one, provided that at least one machine in the new time period has enough available capacity to handle this new job. Fig. 4 illustrates the move for the small example depicted in Fig. 3, which consists in moving job 8 from period 2 to period 3.
- $\mathcal{N}_2$: *Swapping two jobs from different time periods*
  The neighbors of a solution S are obtained by swapping two jobs which are assigned to different time periods, regardless of the machines on which these jobs have been assigned. Fig. 5 shows a possible neighbor of the solution illustrated in Fig. 3, which is obtained by swapping jobs 1 and 12 from different time periods.
- $\mathcal{N}_3$: *Swapping two sequences of jobs from different time periods*
  The neighbors of a solution S are obtained by swapping two sequences of jobs which are assigned to different time periods, regardless of the machines on which these jobs have been assigned. Since only feasible solutions are explored, it is only possible to swap the two sequences if all the jobs are not processed before

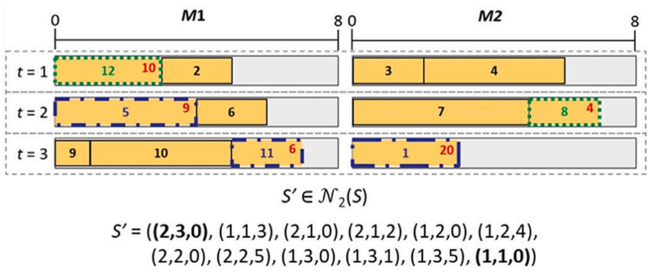

M1
M2
t = 1
t = 2
t = 3
S′ ∈ N2(S)
S′ = ((2,3,0), (1,1,3), (2,1,0), (2,1,2), (1,2,0), (1,2,4), (2,2,0), (2,2,5), (1,3,0), (1,3,1), (1,3,5), (1,1,0))

Fig. 5. Example of a neighbor $S' \in \mathcal{N}_2(S)$ of the solution S (as presented in Fig. 3).

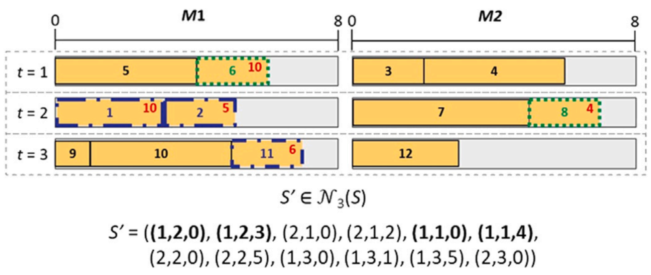

M1
M2
t = 1
t = 2
t = 3
S′ ∈ N3(S)
S′ = ((1,2,0), (1,2,3), (2,1,0), (2,1,2), (1,1,0), (1,1,4), (2,2,0), (2,2,5), (1,3,0), (1,3,1), (1,3,5), (2,3,0))

Fig. 6. Example of a neighbor $S' \in \mathcal{N}_3(S)$ of the solution S (as presented in Fig. 3).

their release dates. For instance, in Example 2.1, it is not possible to swap all the jobs assigned to machine $M1$ at time period $t = 2$ with all jobs from $t = 3$ of the same machine, since the release date of job 10 is 3. An example of a possible neighbor solution is presented in Fig. 6. It is obtained by swapping all the jobs assigned to machine $M1$ at time periods $t = 1$ and $t = 2$.

- $\mathcal{N}_4$: *Swapping two jobs in different machines at the same time period, and inserting a third job*
  The neighbors of a solution S are obtained by swapping two jobs with different processing times and assigned to different machines at the same time period, and by inserting a third job (from a different time period) in the time period of the swapped jobs. For instance, a neighbor of the solution depicted in Fig. 3 may be obtained by swapping jobs 2 and 4 at time period 1, and by inserting job 5 at the same time period (Fig. 7). Note that only the insertion has an impact on the value of the solution. The objective is to explore the possibility of an increase of available capacity at a given time period and machine.
- $\mathcal{N}_5$: *Swapping two jobs at adjacent time periods*
  The neighbors of a solution S are obtained by swapping two jobs which have been assigned to consecutive time periods. In this sense, it consists in a restricted version of $\mathcal{N}_2$, which limits the swaps to the jobs at a maximum distance of one time period. In our implementation, this neighborhood will be used only at the shaking phase, whereas $\mathcal{N}_2$ will be applied during the local search, since it covers naturally a wider solution space ($\mathcal{N}_5 \subset \mathcal{N}_2$). A neighbor of S (Fig. 3) in $\mathcal{N}_5$ may be reached by swapping job 2 at time period 1 with job 5 at time period 2 (Fig. 8).
- $\mathcal{N}_6$: *Swapping two jobs assigned to different machines in the same time period*
  A neighbor solution S' of the solution S is obtained by swapping two jobs with different processing times, but assigned to different machines at the same time period. Therefore, at this given time period, a neighbor solution inevitably has a different job order, which may eventually yield a larger available capacity, as shown in Fig. 9. In this case, the neighbor solution is obtained by swapping jobs 2 and 4. Note that the available capacity increases: in S, at time period 1, we can process at least one more job with processing time up to 3 units, whereas in S', this value increases up to 4 units.
  Since there is no variation on the value of the solution, $\mathcal{N}_6$ is only applied at the shaking phase. Even though, and as in $\mathcal{N}_4$, the increase of available capacity may be profitable in further iterations of the VNS algorithms.

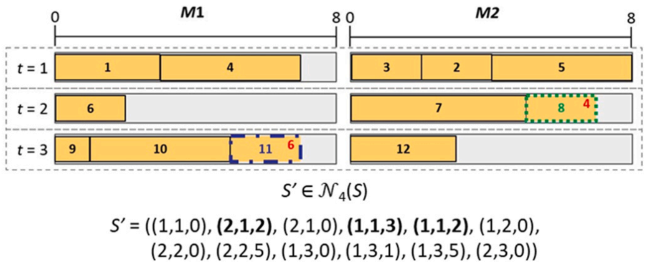

M1
M2
t = 1
t = 2
t = 3
S′ ∈ N4(S)
S′ = ((1,1,0), (2,1,2), (2,1,0), (1,1,3), (1,1,2), (1,2,0), (2,2,0), (2,2,5), (1,3,0), (1,3,1), (1,3,5), (2,3,0))

Fig. 7. Example of a neighbor $S' \in \mathcal{N}_4(S)$ of the solution S (as presented in Fig. 3).

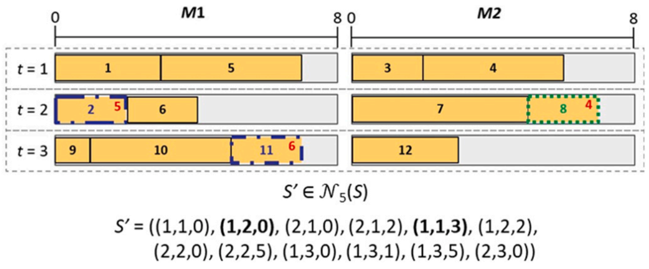

M1
M2
t = 1
t = 2
t = 3
S′ ∈ N5(S)
S′ = ((1,1,0), (1,2,0), (2,1,0), (2,1,2), (1,1,3), (1,2,2), (2,2,0), (2,2,5), (1,3,0), (1,3,1), (1,3,5), (2,3,0))

Fig. 8. Example of a neighbor $S' \in \mathcal{N}_5(S)$ of the solution S (as presented in Fig. 3).

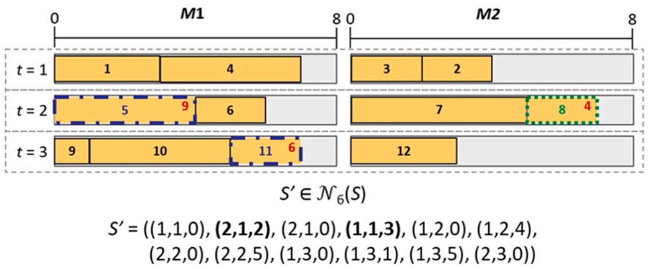

M1
M2
t = 1
t = 2
t = 3
S′ ∈ N6(S)
S′ = ((1,1,0), (2,1,2), (2,1,0), (1,1,3), (1,2,0), (1,2,4), (2,2,0), (2,2,5), (1,3,0), (1,3,1), (1,3,5), (2,3,0))

Fig. 9. Example of a neighbor $S' \in \mathcal{N}_6(S)$ of the solution S (as presented in Fig. 3).

3.4. Variable neighborhood search algorithms

In this section, we describe the three variants of the VNS algorithm that we considered in our implementation. In all the algorithms, we resorted to the constructive heuristic described in Section 3.2 to generate the first feasible solution. From this solution, we explore the neighborhood spaces through different approaches, both at the shaking and local search phases. The shaking phase relies on the same procedures for all the three algorithms. In contrast, according to the variant of the VNS, the local search procedures may vary. A first improvement strategy is used in all variants, *i.e.*whenever a better solution is found, it becomes the new incumbent solution, and it is used at least in the next iteration of the corresponding VNS algorithm. The three VNS algorithms have a stopping criterion given by the maximum execution time (t_{max}), or

Algorithm 2: Basic VNS Algorithm

```
Input:
    Instance for IPSP and a feasible solution $S$;
    Set of neighborhood structures $\mathcal{N} = \{\mathcal{N}_1, \mathcal{N}_2, \mathcal{N}_3, \mathcal{N}_4, \mathcal{N}_5, \mathcal{N}_6\}$ and probabilities $P_{\mathcal{N}_{i\in\{1,2,3,5,6\}}}$;
    Set limit $t_{max} \in \mathbb{Q}$ to total computing time and $k_{max} \in \mathbb{N}$ the maximum of consecutive moves from neighborhood;
Output:
    A feasible and possibly improved solution $S$;
Auxiliary:
    $k \in \mathbb{N}$ and $t$, $t_{start} \in \mathbb{Q}$;

Function BasicVNS($S$, $k_{max}$, $t_{max}$, $\mathcal{N}$, $P$)
1  $t_{start}$ := CPUTime(); $t$ := 0;
2  while $t \le t_{max}$ and $z$(S) > 0 do
3      $k$ := 1;
4      while $k \le k_{max}$ and $z$(S) > 0 do
5          $S'$ := $Shake(S, \mathcal{N}_{i\in\{1,2,3,5,6\}}, P_{\mathcal{N}_{i\in\{1,2,3,5,6\}}}, k)$;      /* shaking phase */
6          $i'$ := $SetI(i)$;      /* setting the neighborhood structure */
7          $S''$ := $FirstImprovement(S', \mathcal{N}_{i'})$;      /* local search phase */
8          if $z(S'') < z(S)$ then
9              $S$ := $S''$;
10             $k$ := 1;
11         end
12         else
13             $k$ := $k$ + 1;
14         end
15     end
16     $t$ := CPUTime() − $t_{start}$;
17 end
18 return $S$
```

by the fact that an incumbent was found with a value 0. In this case, clearly, the incumbent is an optimal solution, since no penalty applies. However, the opposite may not be true, since the optimal solution of a given instance may have penalties. The entire description of the three algorithms, including the details of the shaking and local search phases, is provided below.

3.4.1. Basic variable neighborhood search

Our first approach is based on the Basic Variable Neighborhood Search (BVNS) algorithm introduced in Mladenović and Hansen (1997) which is summarized in Algorithm 2. In the first phase, a given solution S is perturbed through an intensified shaking (Line 5). A neighborhood structure $\mathcal{N}_i$ is selected according to a given probability $P_{\mathcal{N}_i\in\{1,2,3,5,6\}}$, and thus a random jump is performed in the kth neighborhood (k consecutive movements). The maximum number of consecutive movements is limited to k_{max}. The objective is to reach a solution S' that is significantly different from the current to avoid being trapped in a local optimum.

The local search procedure is applied right after to this solution S' using the same neighborhood structure, unless if $i \in \{5,6\}$ since these neighborhoods are only explored in the shaking phase (as referred to in Section 3.3). For this reason, if $i = 5$ or $i = 6$, then the local search resorts to the neighborhoods $\mathcal{N}_2$ or $\mathcal{N}_4$, respectively. This procedure is defined as `SetI(i)` (Line 6). If the obtained solution S'' is better than S, k is set to 1, and S is updated. Otherwise, k is set to $k+1$. In both cases, the process is repeated until a stopping condition is met.

3.4.2. General variable neighborhood search

In this section, we propose a General VNS based algorithm (Hansen et al., 2017) (hereafter denoted by GVNS) to tackle the IPSP. The main difference between BVNS (Section 3.4.1) and GVNS is that the local search in the latter is conducted through the Variable Neighborhood Descent (VND) method (Hansen et al., 2017), which iteratively and sequentially explores the defined neighborhood structures until there is any better solution considering all neighborhood structures. GVNS is summarized in Algorithm 3.

In the VND phase, four neighborhood structures are considered ($\mathcal{N}_{i\in\{1,2,3,4\}}$), through lexicographical order, and a first improvement policy is adopted. A sequential neighborhood change is used, and thus if there is an improvement in the solution, l takes the value 1 (Line 11), and it is incremented otherwise (Line 14) resulting in the selection of a different neighborhood structure. As referred to above, the shaking phase of GVNS is similar to BVNS.

3.4.3. Skewed general variable neighborhood search

Our third approach is a skewed version of the GVNS (Hansen et al., 2017), hereafter denoted by SGVNS and summarized in Algorithm 4. This variant has proved to be effective in different problems (Macedo et al., 2015; Pinto et al., 2020). The shaking and local search phases rely on the same principles as in GVNS. The main difference is that worse solutions can be accepted in the local search phase based on the assumption that the solution is quite different from the current one, while the best solution obtained so far is kept. The difference between two solutions S and S'' can be measured through a distance function $\rho(S,S'')$ with codomain [0,1], which computes the ratio between the number of jobs that are not assigned to the same period in both solutions and the total number of jobs. Let S_{p^i} and S''_{p^i} be the time periods to which job i is assigned in S and S'', respectively. Additionally, $\sigma(S_{p^i}, S''_{p^i})$ take the value 1 if a given job i is assigned to different time periods in the two solutions, and 0 otherwise:

$$\sigma(S_{p^i}, S''_{p^i}) = \begin{cases} 1, & \text{if} S_{p^i} \neq S''_{p^i} \\ 0, & \text{if} S_{p^i} = S''_{p^i} \end{cases}. \tag{8}$$

Therefore, the distance function can be defined as follows:

$$\rho(S,S'') = \frac{\sum_{i=1}^{|N|} \sigma(S_{p^i}, S''_{p^i})}{|N|}. \tag{9}$$

Algorithm 3: General VNS Algorithm

```
Input:
    Instance for IPSP and a feasible solution $S$;
    Set of neighborhood structures $\mathcal{N} = \{\mathcal{N}_1, \mathcal{N}_2, \mathcal{N}_3, \mathcal{N}_4, \mathcal{N}_5, \mathcal{N}_6\}$ and probabilities $P_{\mathcal{N}_{i\in\{1,2,3,5,6\}}}$;
    Set limit $t_{max} \in \mathbb{Q}$ to total computing time, $k_{max} \in \mathbb{N}$ the maximum of consecutive moves from neighborhood and $l_{max} \in \mathbb{N}$ the number
    of the neighborhood structures consider to local search phase;
Output:
    A feasible and possibly improved solution $S$;
Auxiliary:
    $k, l \in \mathbb{N}$ and $t$, $t_{start} \in \mathbb{Q}$;

Function GeneralVNS($S$, $k_{max}$, $t_{max}$, $\mathcal{N}$, $P$)
1  $t_{start}$ := CPUTime(); $t$ := 0;
2  while $t \le t_{max}$ and $z(S) > 0$ do
3      $k$ := 1;
4      while $k \le k_{max}$ and $z(S) > 0$ do
5          $S'$ := $Shake(S, \mathcal{N}_{i\in\{1,2,3,5,6\}}, P_{\mathcal{N}_{i\in\{1,2,3,5,6\}}}, k)$;        /* shaking phase */
6          $l$ := 1 ;        /* local search phase using $VND(S', 4, \mathcal{N}_{i\in\{1,2,3,4\}})$ */
7          while $l \le 4$ and $z(S') > 0$ do
8              $S'''$ := $FirstImprovement(S'', \mathcal{N}_l\})$;
9              if $z(S''') < z(S')$ then
10                 $S'$ := $S'''$;
11                 $l$ := 1;
12             end
13             else
14                 $l$ := $l + 1$;
15             end
16         end
17         if $z(S') < z(S)$ then
18             $S$ := $S'$;
19             $k$ := 1;
20         end
21         else
22             $k$ := $k + 1$;
23         end
24     end
25     $t$ := CPUTime() − $t_{start}$;
26 end
27 return $S$
```

As in Macedo et al. (2015) and in Pinto et al. (2020), a given solution S'' is accepted if

$$z(S'') < (1 + \alpha\rho(S, S''))z(S), \tag{10}$$

with $\alpha \in [0, 1]$ (Line 15).

The value of α may not be the same during all the execution of the algorithm. In fact, in very particular cases, two given solutions may be significantly different while their value in the objective function is very close. Generally, these situations happen for particular instances with a large number of jobs sharing the same attributes as the due dates or penalty factors. In these situations, a cycle may occur, preventing the algorithm from exploring all the neighborhood structures, and thus not achieving the stopping condition. In order to avoid these situations, a parameter of tolerance time (*tolerance*) is defined. If the maximum computing time plus the tolerance time is reached, then the value of α is set to 0 (as can be seen in Lines 12–13). Thus, a worse solution will be rejected, and it is possible to establish a limit for the execution of the algorithm, returning the best solution obtained so far (Line 24).

4. Computational experiments

4.1. Benchmark instances

To assess the quality of the approaches described above, we performed extensive computational experiments using benchmark instances proposed for the IPSP in Kis and Kovács (2012). In all instances, the time horizon is divided into identical periods, each one with the same length ($P = 100$). The instances are divided into three sets (A, B, and C), according to the processing time of the jobs. Thus, the set A is composed of instances whose jobs have a short processing time.

$$1 \le p_j \le 33, \forall j \in N, |N| \in \{100, 150, 200, 250, 300\},$$

The set C is only composed by instances whose jobs have a large processing time:

$$34 \le p_j \le 100, \forall j \in N, |N| \in \{40, 60, 80, 100\}.$$

In set B, half of the jobs have a short processing time, and the other half has a large processing time, with $|N| \in \{50, 100, 150, 200, 250, 300\}$. Each instance has up to 10 machines $|M| \in \{2, 6, 10\}$. The release and due dates correspond to time periods within the time window [0, τ_0], with $\tau_0 \in \{2, 6, 10\}$ (τ_0 is the so-called nominal length of the time horizon).

To ensure a feasible solution for each instance, the authors considered a larger time horizon τ computed as follows:

$$\tau = \tau_0 + \left\lceil 2 \times \sum_{j=1}^{N} \frac{p_j}{(P \times M)} \right\rceil . \tag{11}$$

For each combination of $|N|$, $|M|$, and τ_0, five different instances were generated. As a result, the total number of instances is 675 (225, 270, and 180 for sets A, B, and C, respectively).

Algorithm 4: Skewed General VNS Algorithm

```
Input:
    Instance for IPSP and a feasible solution $S$;
    Set of neighborhood structures $\mathcal{N} = \{\mathcal{N}_1, \mathcal{N}_2, \mathcal{N}_3, \mathcal{N}_4, \mathcal{N}_5, \mathcal{N}_6\}$ and probabilities $P_{\mathcal{N}_{i\in\{1,2,3,5,6\}}}$;
    Set limit $t_{max} \in \mathbb{Q}$ to total computing time, set $k_{max} \in \mathbb{N}$ the maximum of consecutive moves from neighborhood, define the parameter
    $\alpha \in \mathbb{Q}$ for skewed moves, a distance function $\rho$ between two solutions and $tolerance \in \mathbb{Q}$ time possible to beyond $t_{max}$;
Output:
    A feasible and possibly improved solution $S$;
Auxiliary:
    $k \in \mathbb{N}$; $t$, $t_{start} \in \mathbb{Q}$ and solution $S_{best}$;

Function SkewedGeneralVNS($S$, $k_{max}$, $t_{max}$, $\mathcal{N}$, $P$, $\alpha$, $tolerance$)
t_start := CPUTime(); t := 0;
while $t \leq t_{max}$ and $z(S) > 0$ do
    $k := 1$;
    $S_{best} := S$;
    while $k \leq k_{max}$ and $z(S) > 0$ do
        $S' := Shake(S, \mathcal{N}_{i\in\{1,2,3,5,6\}}, P_{\mathcal{N}_{i\in\{1,2,3,5,6\}}}, k)$;        /* shaking phase */
        $S'' := VND(S', 4, \mathcal{N}_{i\in\{1,2,3,4\}})$;        /* local search phase */
        if $z(S'') < z(S_{best})$ then
            $S_{best} := S''$;
        end
        $t :=$ CPUTime() $- t_{start}$;
        if $t > t_{max} + tolerance$ then
            $\alpha := 0$;
        end
        if $z(S'') < (1 + \alpha\rho(S, S''))z(S)$ then
            $S := S''$;
            $k := 1$;
        end
        else
            $k := k + 1$;
        end
    end
    $t :=$ CPUTime() $- t_{start}$;
    $S := S_{best}$;
end
return $S$
```

4.2. Preliminary computational tests

Preliminary statistical tests based on the multiple comparison procedure were conducted to set the values of parameters using the BVNS algorithm in the set of instances A. However, the parameters were kept in the other VNS variants and in the other sets for the sake of consistency in the comparisons. The impact of variations in the following parameters was evaluated:

- the number of used neighborhood structures (ns ∈ {2, 4, 6});
- the maximum execution time (t_{max} ∈ {1, 5, 9} in seconds);
- the maximum value of consecutive moves (k_{max} ∈ {10, 15, 20, 25, 30, 40}).

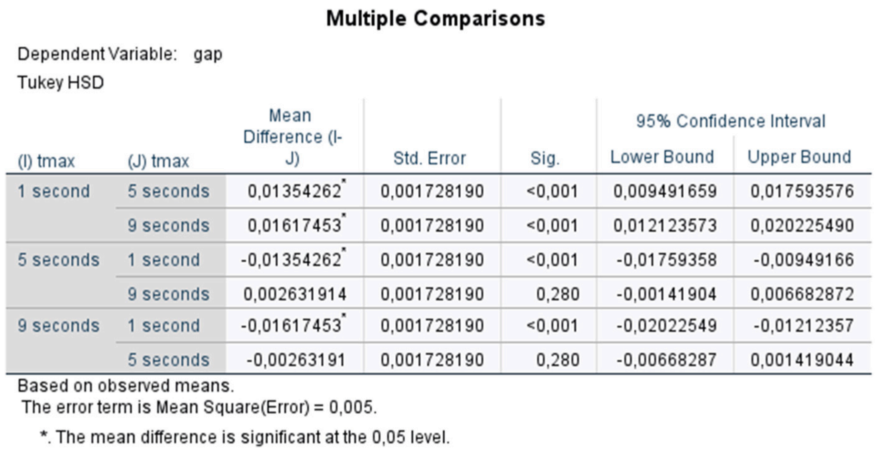
Firstly, for each combination of parameters between ns ∈ {2, 4, 6} and t_{max} ∈ {1, 5, 9}, five replications were performed for each instance of set A (225 instances) with k_{max}=20. We evaluated the results obtained using Tukey's HSD test with the *gap* as the dependent variable, which corresponds to the average optimality gap, given by (best upper bound - best lower bound)/(best upper bound). In other words, the *gap* can be understood as the difference between our results and the lower bound, divided by the same lower bound (to observe the proportion of this difference).

Concerning the maximum execution time of the algorithm (t_{max}), as t_{max} is increased, the average *gap* decreases (*i.e.*, longer execution times reach solutions closer to the optimal ones). Considering t_{max}=1, statistically significant differences are obtained when compared to t_{max}=5 and t_{max}=9. However, as the differences between these two last values are not statically significant (Fig. 10), the time limit was set to 5 seconds to reduce the overall computational execution time for all instances and all replications while keeping the efficiency of the proposed algorithm.

Regarding the number of neighborhood structures (ns), it is possible to verify that the inclusion of more (and new) neighborhood structures led to a decrease in the *gap* (Fig. 11). The statistical differences between having or not having the 6 proposed neighborhood structures are significant (when compared to ns=2 or ns=4). Thus, based on these observations, we consider ns=6 and t_{max}=5 (seconds).

Subsequently, the impact of the maximum value of consecutive movements was assessed in the shaking phase (k_{max}), considering ns=6 and t_{max}=5 (seconds). Similarly, we performed 5 runs (5 × 225 instances) for each combination of k_{max} ∈ {10, 15, 20, 25, 30, 40}. Again, using the *gap* as the dependent variable, the results showed no statistically significant differences between different combinations with a significance level of 5% (Fig. 12). However, it is possible to identify that k_{max}=20 will be the minimum value of the mean *gap*. On the contrary, k_{max}=10 and k_{max}=40 reach worst values. Therefore, k_{max} was set to 20 since the closest solutions to the globally optimal ones were obtained.

Concerning the different probabilities for selecting a given neighborhood structure, and after several computational experiments, it



Multiple Comparisons

Dependent Variable: gap

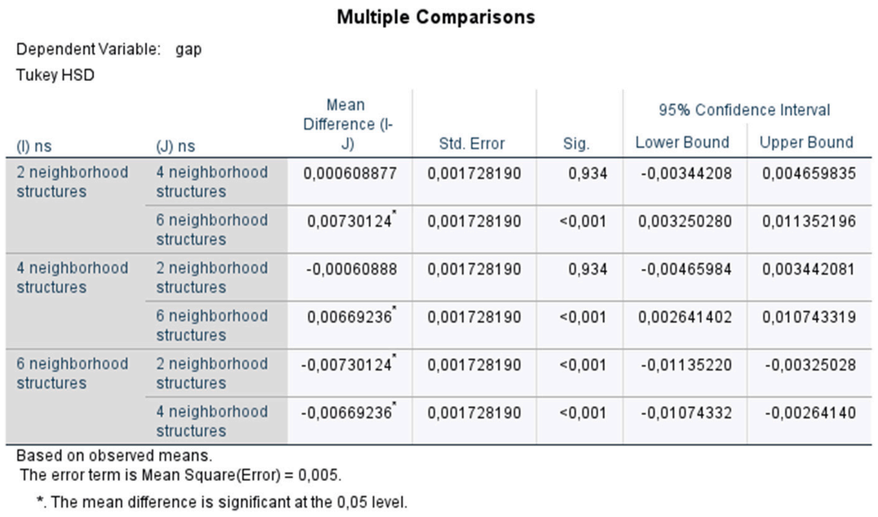
Tukey HSD

(I) tmax	(J) tmax	Mean Difference (I-J)	Std. Error	Sig.	95% Confidence Interval Lower Bound	95% Confidence Interval Upper Bound
1 second	5 seconds	0,01354262*	0,001728190	<0,001	0,009491659	0,017593576
	9 seconds	0,01617453*	0,001728190	<0,001	0,012123573	0,020225490
5 seconds	1 second	-0,01354262*	0,001728190	<0,001	-0,01759358	-0,00949166
	9 seconds	0,002631914	0,001728190	0,280	-0,00141904	0,006682872
9 seconds	1 second	-0,01617453*	0,001728190	<0,001	-0,02022549	-0,01212357
	5 seconds	-0,00263191	0,001728190	0,280	-0,00668287	0,001419044

Based on observed means.
The error term is Mean Square(Error) = 0,005.

*. The mean difference is significant at the 0,05 level.

Fig. 10. Multiple comparisons between different maximum execution times (t_{max}).



Multiple Comparisons

Dependent Variable: gap

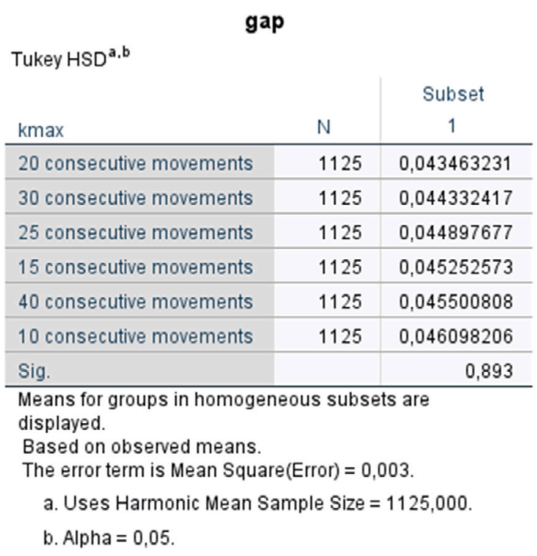
Tukey HSD

(I) ns	(J) ns	Mean Difference (I-J)	Std. Error	Sig.	95% Confidence Interval Lower Bound	95% Confidence Interval Upper Bound
2 neighborhood structures	4 neighborhood structures	0,000608877	0,001728190	0,934	-0,00344208	0,004659835
	6 neighborhood structures	0,00730124*	0,001728190	<0,001	0,003250280	0,011352196
4 neighborhood structures	2 neighborhood structures	-0,00060888	0,001728190	0,934	-0,00465984	0,003442081
	6 neighborhood structures	0,00669236*	0,001728190	<0,001	0,002641402	0,010743319
6 neighborhood structures	2 neighborhood structures	-0,00730124*	0,001728190	<0,001	-0,01135220	-0,00325028
	4 neighborhood structures	-0,00669236*	0,001728190	<0,001	-0,01074332	-0,00264140

Based on observed means.
The error term is Mean Square(Error) = 0,005.

*. The mean difference is significant at the 0,05 level.

Fig. 11. Multiple comparisons between different number of neighborhood structures (ns).



gap

Tukey HSD[a,b]

kmax	N	Subset 1
20 consecutive movements	1125	0,043463231
30 consecutive movements	1125	0,044332417
25 consecutive movements	1125	0,044897677
15 consecutive movements	1125	0,045252573
40 consecutive movements	1125	0,045500808
10 consecutive movements	1125	0,046098206
Sig.		0,893

Means for groups in homogeneous subsets are displayed.
Based on observed means.
The error term is Mean Square(Error) = 0,003.

a. Uses Harmonic Mean Sample Size = 1125,000.

b. Alpha = 0,05.

Fig. 12. Homogeneous subset between different values of consecutive movements (k_{max}).

became clear that some structures spend more time than others. Thus, in general, the algorithm tends to favor the least time-consuming ones (by giving them a higher probability) while keeping the possibility of exploring a broader solution space.

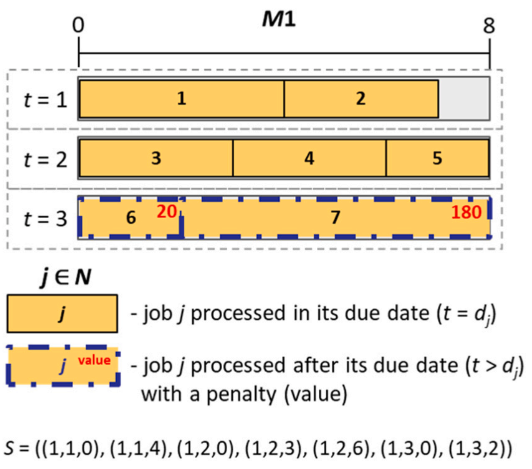
For the SGVNS variant, the values of α and tolerance time (*tolerance*) parameters were obtained experimentally through several computational tests. Values of α less than 0.05 did not allow much diversification of solutions. On the other hand, values greater than 0.05 lead to a large scatter in the solution space that could not converge to better solutions. Therefore, α was set to 0.05, and *tolerance* was set to 2 seconds.

To assess the sequence of the performing order of the neighborhood structures, a set of computational tests with different orders and combinations were performed. In the proposed approaches, the order of the neighborhood structures is only relevant in the local search phase using the VND method since, in the remaining cases, the neighborhood structure selection relies on different probabilities.

The VND method (applied both in GVNS and SGVNS) has the neighborhood structures $\mathcal{N}_1$ to $\mathcal{N}_4$ by increasing numerical order.

The structure ($\mathcal{N}_1$) assumes an important role in attempting to fill gaps in the time periods of machines with available time to process jobs. After filling these spaces, it might be useful to perform exchanges between assigned jobs (using $\mathcal{N}_2$) to seek better solutions.

The neighborhood structure $\mathcal{N}_3$ is specially adequate for situations in which jobs have a significant processing time when compared to the



M1
0
8
t = 1
1
2
t = 2
3
4
5
t = 3
6
20
7
180
j ∈ N
j
- job j processed in its due date (t = dj)
j value
- job j processed after its due date (t > dj) with a penalty (value)
S = ((1,1,0), (1,1,4), (1,2,0), (1,2,3), (1,2,6), (1,3,0), (1,3,2))

Fig. 13. Feasible solution S with a total penalty value equal to 200.

Table 2
Attributes for each job $j \in N$.

j	r_j	d_j	p_j	e_j	l_j
1	1	1	4	0	10
2	1	1	3	0	15
3	1	2	3	30	30
4	1	2	3	20	30
5	1	2	2	25	20
6	1	1	2	0	10
7	1	1	6	0	90

capacity of the period time (as it happens, for example, in instances of set B and C).

These situations can arise when there are no machines with available capacity to insert a given job with a large processing time or if it is impossible to exchange it with another job due to the capacity limit. For the sake of clarity, Example 4.1) depicts this situation.

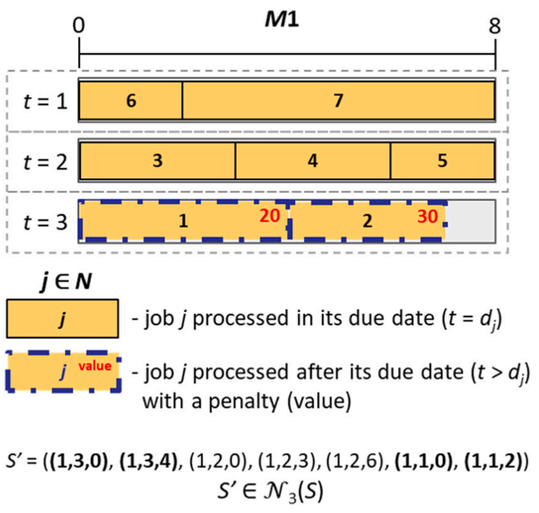
Example 4.1. Considering a instance with 7 jobs ($N = \{1, 2, \ldots, 7\}$) which must be processed in one machine ($M = \{1\}$) in the time horizon of the problem defined as $T = \{1, 2, 3\}$, with $P = 8$ and $\tau = 3$. For the sake of conciseness, without losing generality, a single machine will be used to be more concise, but it remains applicable when increasing the number of machines.

Similar to Table 1, Table 2 lists all the parameters associated with all jobs. Processing times are given in hours for each job, and the due and release dates are expressed in time periods (shifts). Fig. 13 depicts a feasible solution S for this instance of the IPSP with a total penalty equal to 200 (due to the penalty of 2 delayed periods for job 6 and job 7, 2×10 and 2×90, respectively).

In the solution S, job 7, which has a high penalty compared to the other jobs, cannot be inserted in a previous period of time (through $\mathcal{N}_1$), nor is it possible to exchange it with another job (through $\mathcal{N}_2$), because the machine has no available time. In other words, job 7 would be prevented from changing its assigned period and, therefore, prevented from reducing its penalty even though it presents a higher penalty value (l_7=90).

However, through $\mathcal{N}_3$ the solution S will be able to be improved. In Fig. 14, swapping the sequence of jobs from one period with the sequence from other results in a lower penalty: exchange between the sequences of jobs from period 1 with period 3 will reduce the penalty from 200 to 50.

Thus, with $\mathcal{N}_3$ we can unblock jobs that are stuck in some periods, from where it would be impossible to move them to other periods (through insertion) or exchange with other jobs (one by one).



M1
0
8
t = 1
6
7
t = 2
3
4
5
t = 3
1
20
2
30
j ∈ N
j
- job j processed in its due date (t = dj)
j value
- job j processed after its due date (t > dj) with a penalty (value)
S′ = ((1,3,0), (1,3,4), (1,2,0), (1,2,3), (1,2,6), (1,1,0), (1,1,2))
S′ ∈ 𝒩3(S)

Fig. 14. Solution S' obtained through $\mathcal{N}_3$ applied in original solution S.

Neighborhood structure $\mathcal{N}_4$ was specially designed for the context of the IPSP. It appears in the last position since it proved to be the most computationally challenging neighborhood structure, and hence it would be profitable if it is less requested. Indeed, the search is restarted in the first neighborhood structure if the incumbent solution is improved in any neighborhood structure. On the other hand, if the solution is not improved does not improve, the search continues to the following neighborhood structure (according to the defined order).

As referred to above, structures $\mathcal{N}_3$ and $\mathcal{N}_4$ are helpful in particular contexts, and they can become very time-consuming, which may not necessarily be reflected in solution improvements. Thus, they present a lower probability of being chosen in BVNS Algorithm (Algorithm 2).

4.3. Computational results and discussion

In this section, we report on the computational results obtained for the benchmark instances described above by all the proposed approaches. Computational experiments were performed on a PC with an Intel 8th Gen Core i7-8565 processor and 16 GB of RAM.

According to the insights of the preliminary tests reported in Section 4.2, the computational time limit (t_{max}) was set to 5 seconds for all approaches, and the maximum number of consecutive movements (k_{max}) was set to 20. The number of used neighborhood structures was 6, and the probability of selecting each one in the shaking phase in all algorithms was set as follows: $P_{\mathcal{N}_1} = 0.25$, $P_{\mathcal{N}_2} = 0.25$, $P_{\mathcal{N}_3} = 0.15$, $P_{\mathcal{N}_4} = 0.00$, $P_{\mathcal{N}_5} = 0.25$ and $P_{\mathcal{N}_6} = 0.10$. Finally, α=0.05 and $tolerance$=2 seconds.

Due to the random nature of these procedures, five runs were performed for each instance and each proposed algorithm. The best results obtained over the 5 runs for each algorithm were compared with the best results achieved with the different exact approaches proposed in Rietz et al. (2016). Note that, although lower and upper bounds were provided in Rietz et al. (2016) with a time limit of 1200 seconds for each run, in Rietz et al. (2016), these authors report only the results for instances with small values of $|N|$ as in Kis and Kovács (2012). The authors provided us the unpublished results for the large-size instances. Therefore, our comparison will include the whole set of benchmark instances. For the sake of clarity, we divided the instances into a set of so-called small-size instances whose results were reported in Kis and Kovács (2012) and Rietz et al. (2016), and into a set of large-size instances that includes all the other instances. As in Kis and Kovács (2012) and Rietz et al. (2016), the results are presented for groups of 15 instances according to the set and the values of $|N|$ and $|M|$ (Tables 3 and 4). Tables 5 and 6 propose a different perspective of the results for groups from the same set and with the same values of $|N|$ and τ_0.

Table 3
Computational results by number of jobs and machines for the small-size instances.

set	*dim*	$\lvert N\rvert$	$\lvert M\rvert$	BVNS			GVNS			SGVNS		
				opt	*gap*	*t*	*opt*	*gap*	*t*	*opt*	*gap*	*t*
A	SI	100	2	0	0.061	2.110	0	0.048	2.450	0	0.049	2.824
		100	6	12	0.004	0.444	14	0.003	0.573	14	0.003	0.372
		100	10	14	0.001	0.196	14	0.001	0.170	14	0.001	0.024
A	SI	150	2	0	0.042	2.981	0	0.030	3.015	0	0.028	4.012
		150	6	5	0.021	1.352	8	0.009	1.311	8	0.009	1.007
		150	10	11	0.007	0.291	12	0.003	0.309	11	0.003	0.221
A	SI	200	2	0	0.039	3.674	0	0.027	4.086	0	0.032	4.350
		200	6	3	0.079	1.808	5	0.029	1.721	5	0.028	2.243
		200	10	10	0.009	0.865	10	0.005	1.003	10	0.004	1.111
total/avg.		***set A - SI***		**55**	**0.029**	**1.525**	**63**	**0.017**	**1.626**	**62**	**0.017**	**1.796**
B	SI	50	2	4	0.046	1.247	12	0.041	1.069	15	0.040	0.895
		50	6	11	0.023	0.246	15	0.019	0.119	15	0.019	0.479
		50	10	14	0.009	0.275	15	0.007	0.034	14	0.009	0.040
B	SI	100	2	0	0.045	1.747	0	0.024	3.068	0	0.023	3.808
		100	6	1	0.095	1.502	4	0.047	1.892	6	0.039	2.307
		100	10	6	0.057	1.356	7	0.038	0.707	8	0.035	1.528
B	SI	150	2	0	0.035	3.271	0	0.019	3.897	0	0.028	4.533
		150	6	0	0.080	3.267	0	0.048	2.440	0	0.044	4.061
		150	10	1	0.113	2.025	3	0.059	1.816	5	0.047	2.509
total/avg.		***set B - SI***		**37**	**0.056**	**1.660**	**56**	**0.034**	**1.671**	**63**	**0.032**	**2.240**
C	SI	40	2	12	0.000	0.329	15	0.000	0.012	15	0.000	0.047
		40	6	15	0.000	0.244	15	0.000	0.006	15	0.000	0.014
		40	10	15	0.000	0.040	15	0.000	0.002	15	0.000	0.003
C	SI	60	2	11	0.002	0.785	15	0.000	0.211	15	0.000	0.652
		60	6	14	0.001	0.408	15	0.000	0.079	15	0.000	0.132
		60	10	14	0.000	0.490	15	0.000	0.084	15	0.000	0.057
C	SI	80	2	9	0.001	0.683	15	0.000	0.908	15	0.000	2.149
		80	6	10	0.002	1.020	15	0.000	0.512	15	0.000	1.329
		80	10	11	0.005	1.186	15	0.000	0.637	15	0.000	0.202
total/avg.		***set C - SI***		**111**	**0.001**	**0.576**	**135**	**0.000**	**0.272**	**135**	**0.000**	**0.510**

The results for the small and large size instances are presented in Tables 3 and 5, and Tables 4 and 6, respectively. A summary of the results is provided in Table 7 including the best results through exact methods provided by Rietz et al. (2016). The meaning of each column is as follows:

- *set*: set of instances;
- *dim*: small-size instances (SI) or large-size instances (LI) according to $|N|$;
- $|N|$: number of jobs;
- $|M|$: number of machines;
- τ_0: nominal length of the time horizon;
- *opt*: number of instances solved up to optimality;
- *gap*: average optimality gap, given by (best upper bound - best lower bound)/(best upper bound);
- *t*: average computing time to find the solution (in seconds).

From the results obtained, and considering the whole set of instances (small- and large-size), the GVNS and SGVNS algorithm appear to perform better than BVNS. In both cases, the number of optimal solutions found is larger, and the average optimality gap remains smaller than with BVNS. Among the three approaches, SGVNS is the algorithm that requires the highest average computational time to reach the final solution. Nevertheless, the computational times remain very low for all the tested algorithms. Comparing GVNS with SGVNS, we observe that the former performs slightly better for sets A and C not only in the number of optimal solutions (75 against 74 for set A, and 180 against 179 for set C), but also in the value of the average optimality gap. For set B, despite the identical value of the optimality gap, SGVNS outperforms GVNS in terms of optimal solutions found (63 against 57).

Figs. 15, 16 and 17 provide, respectively, a comparison of the average optimality gap, computational times and instances solved up to optimality by BVNS, GVNS, SGVNS and the exact methods described in Rietz et al. (2016) (denoted as UB). Additionally, in Table 7, the total and average results of Tables 3–6 are summarized for all the instances grouped by set and size and compared with the results of the exact methods described in Rietz et al. (2016).

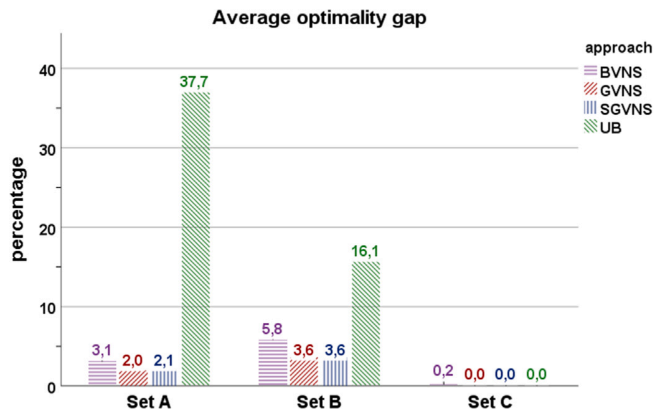


Fig. 15. Average optimality gap (*gap*) for each set and each approach.

In set A, 75 optimal solutions are obtained through GVNS, whereas the exact methods (UB) reached 83. However, the average optimality gaps achieved by all our VNS algorithms are significantly lower. Additionally, for the small-size instances, the solutions obtained by GVNS are on average 21% closer to the lower bound than the solutions obtained through exact approaches. For larger instances, this difference is even more significant: GVNS achieves an average optimality gap of 2.5% against the 60% of the exact methods. Another major benefit of all

Table 4
Computational results by number of jobs and machines for the large-size instances.

set	*dim*	$\|N\|$	$\|M\|$	BVNS			GVNS			SGVNS		
				opt	*gap*	*t*	*opt*	*gap*	*t*	*opt*	*gap*	*t*
A	LI	250	2	0	0.032	3.210	0	0.025	3.980	0	0.030	4.301
		250	6	0	0.046	2.881	0	0.028	3.299	0	0.027	4.165
		250	10	7	0.022	1.641	7	0.022	1.620	7	0.018	1.890
A	LI	300	2	0	0.025	4.189	0	0.021	5.033	0	0.027	5.509
		300	6	0	0.052	3.717	0	0.034	3.430	0	0.035	4.210
		300	10	5	0.029	2.636	5	0.016	2.539	5	0.017	2.568
total/avg.		***set A - LI***		**12**	**0.034**	**3.046**	**12**	**0.025**	**3.317**	**12**	**0.026**	**3.774**
B	LI	200	2	0	0.031	3.682	0	0.019	3.821	0	0.028	5.983
		200	6	0	0.060	3.917	0	0.037	3.793	0	0.036	4.910
		200	10	0	0.110	2.437	0	0.072	3.057	0	0.074	3.505
B	LI	250	2	0	0.029	3.995	0	0.019	5.046	0	0.025	8.968
		250	6	0	0.056	3.219	0	0.037	4.801	0	0.037	6.846
		250	10	0	0.087	2.847	0	0.056	4.442	0	0.052	5.830
B	LI	300	2	0	0.030	4.306	0	0.019	7.947	0	0.022	11.442
		300	6	0	0.056	4.220	0	0.034	5.153	0	0.035	8.651
		300	10	0	0.085	4.276	0	0.050	4.768	0	0.046	6.552
total/avg.		***set B - LI***		**0**	**0.060**	**3.655**	**0**	**0.038**	**4.759**	**0**	**0.039**	**6.965**
C	LI	100	2	12	0.000	1.350	15	0.000	0.551	14	0.000	2.064
		100	6	6	0.004	1.143	15	0.000	1.220	15	0.000	1.445
		100	10	10	0.003	0.651	15	0.000	0.334	15	0.000	0.807
total/avg.		***set C - LI***		**28**	**0.002**	**1.048**	**45**	**0.000**	**0.702**	**44**	**0.000**	**1.439**

Table 5
Computational results by number of jobs and τ_0 for the small-size instances.

set	*dim*	$\|N\|$	τ_0	BVNS			GVNS			SGVNS		
				opt	*gap*	*t*	*opt*	*gap*	*t*	*opt*	*gap*	*t*
A	SI	100	2	6	0.011	1.521	8	0.008	1.988	8	0.009	1.802
		100	6	10	0.018	0.553	10	0.014	0.968	10	0.014	0.846
		100	10	10	0.037	0.676	10	0.031	0.237	10	0.031	0.573
A	SI	150	2	1	0.018	2.447	2	0.010	2.191	1	0.012	2.256
		150	6	5	0.030	1.505	8	0.016	1.642	8	0.015	1.739
		150	10	10	0.022	0.672	10	0.016	0.802	10	0.012	1.245
A	SI	200	2	0	0.020	2.865	0	0.012	3.713	0	0.013	3.351
		200	6	5	0.030	2.080	5	0.018	1.760	5	0.020	2.548
		200	10	8	0.078	1.402	10	0.030	1.336	10	0.032	1.806
total/avg.		***set A - SI***		**55**	**0.029**	**1.525**	**63**	**0.017**	**1.626**	**62**	**0.017**	**1.796**
B	SI	50	2	5	0.028	1.188	13	0.020	0.469	14	0.022	0.906
		50	6	11	0.026	0.321	15	0.024	0.620	15	0.024	0.281
		50	10	13	0.024	0.260	14	0.023	0.133	15	0.022	0.226
B	SI	100	2	0	0.026	1.982	1	0.014	2.048	2	0.013	3.293
		100	6	1	0.099	2.023	2	0.057	2.326	3	0.053	2.445
		100	10	6	0.072	0.601	8	0.037	1.293	9	0.031	1.905
B	SI	150	2	0	0.024	3.280	0	0.010	3.027	0	0.013	3.719
		150	6	0	0.073	2.871	0	0.044	2.740	0	0.042	3.969
		150	10	1	0.131	2.413	3	0.072	2.386	5	0.065	3.415
total/avg.		***set B - SI***		**37**	**0.056**	**1.660**	**56**	**0.034**	**1.671**	**63**	**0.032**	**2.240**
C	SI	40	2	15	0.000	0.281	15	0.000	0.005	15	0.000	0.016
		40	6	13	0.000	0.205	15	0.000	0.007	15	0.000	0.018
		40	10	14	0.000	0.128	15	0.000	0.008	15	0.000	0.030
C	SI	60	2	13	0.000	0.535	15	0.000	0.049	15	0.000	0.276
		60	6	15	0.000	1.043	15	0.000	0.258	15	0.000	0.408
		60	10	11	0.003	0.107	15	0.000	0.068	15	0.000	0.157
C	SI	80	2	11	0.000	0.651	15	0.000	0.142	15	0.000	0.408
		80	6	8	0.003	1.485	15	0.000	1.236	15	0.000	1.766
		80	10	11	0.004	0.752	15	0.000	0.679	15	0.000	1.506
total/avg.		***set C - SI***		**111**	**0.001**	**0.576**	**135**	**0.000**	**0.272**	**135**	**0.000**	**0.510**

the VNS approaches described in this paper can be seen in the execution time since, on average, these methods need only 2.3 seconds to find the solution against the 778.4 seconds needed by the exact methods. In set B, SGVNS appears to be the most effective among the proposed algorithms. For the small-size instances, the average optimality gap achieved with SGVNS is almost the same as with the exact methods. However, SGVNS is able to obtain those results in 2.2 seconds on average against the 460 seconds spent by the exact methods. For the large-size instances, the average optimality gap achieved by GVNS is significantly better than the gaps obtained with the exact methods,

Table 6
Computational results by number of jobs and τ_0 for the large-size instances.

set	*dim*	$\|N\|$	τ_0	BVNS			GVNS			SGVNS		
				opt	*gap*	*t*	*opt*	*gap*	*t*	*opt*	*gap*	*t*
A	LI	250	2	0	0.015	3.534	0	0.012	3.691	0	0.013	4.822
		250	6	2	0.042	2.555	2	0.036	3.196	2	0.036	2.858
		250	10	5	0.043	1.643	5	0.028	2.012	5	0.027	2.676
A	LI	300	2	0	0.018	4.022	0	0.011	4.250	0	0.014	3.467
		300	6	0	0.046	3.673	0	0.030	4.003	0	0.034	4.724
		300	10	5	0.042	2.846	5	0.031	2.750	5	0.031	4.096
total/avg.		***set A - LI***		**12**	**0.034**	**3.046**	**12**	**0.025**	**3.317**	**12**	**0.026**	**3.774**
B	LI	200	2	0	0.022	2.828	0	0.011	3.777	0	0.018	4.040
		200	6	0	0.066	3.958	0	0.041	3.525	0	0.039	5.790
		200	10	0	0.113	3.250	0	0.076	3.369	0	0.081	4.567
B	LI	250	2	0	0.023	3.294	0	0.011	4.802	0	0.018	8.029
		250	6	0	0.057	3.151	0	0.036	4.918	0	0.037	7.307
		250	10	0	0.091	3.616	0	0.064	4.569	0	0.060	6.310
B	LI	300	2	0	0.026	4.182	0	0.013	6.363	0	0.017	9.544
		300	6	0	0.054	4.602	0	0.033	5.758	0	0.032	8.983
		300	10	0	0.091	4.017	0	0.058	5.747	0	0.054	8.118
total/avg.		***set B - LI***		**0**	**0.060**	**3.655**	**0**	**0.038**	**4.759**	**0**	**0.039**	**6.965**
C	LI	100	2	10	0.001	0.295	15	0.000	0.543	15	0.000	0.654
		100	6	10	0.002	1.106	15	0.000	0.973	14	0.000	2.152
		100	10	8	0.004	1.743	15	0.000	0.589	15	0.000	1.509
total/avg.		***set C - LI***		**28**	**0.002**	**1.048**	**45**	**0.000**	**0.702**	**44**	**0.000**	**1.439**

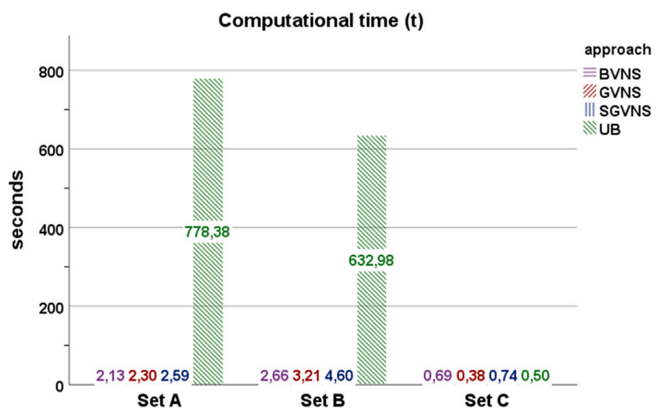


Fig. 16. Average computational time to obtain a final solution (t) for each set and each approach.

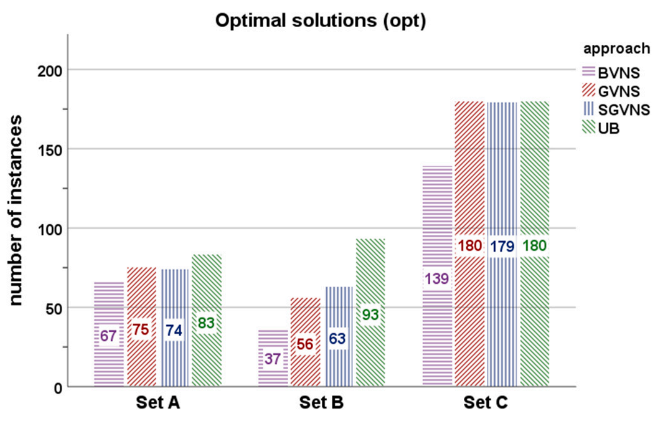


Fig. 17. Number of instances solved up to optimality (opt) for each set and each approach.

while the computing time is shorter by almost 800 seconds. In set C, the GVNS approach was able to obtain an optimal solution for all the instances in a short computational time (0.4 seconds on average), which is clearly competitive with the results obtained in Rietz et al. (2016).

For a detailed analysis, we provide a comparison of the total/average results achieved by all proposed approaches with the ones obtained by Rietz et al. (2016) considering the groups of 15 instances for equal values of $|N|$ and $|M|$ and for equal values of $|N|$ and τ_0.

For set A the groups of instances with an equal value of $|N|$ and τ_0, the average results achieved with the VNS approaches outperform always the best results of Rietz et al. (2016). For set B, the average optimality gap remains barely the same whatever the grouping criterion that is used. The exact methods from the literature give better results for the shortest instances, while our VNS algorithms perform better with the large-size instances. For sets A and B, the difference in computational time among the approaches presented in this paper and the exact methods described in Rietz et al. (2016) is very significant. Regarding the set C, with a maximum number of jobs per instance lower than the one of the other sets ($|N|$=100), and given the attributes of the jobs that allow fewer combinations, the exact methods can also be extremely fast as our VNS approaches.

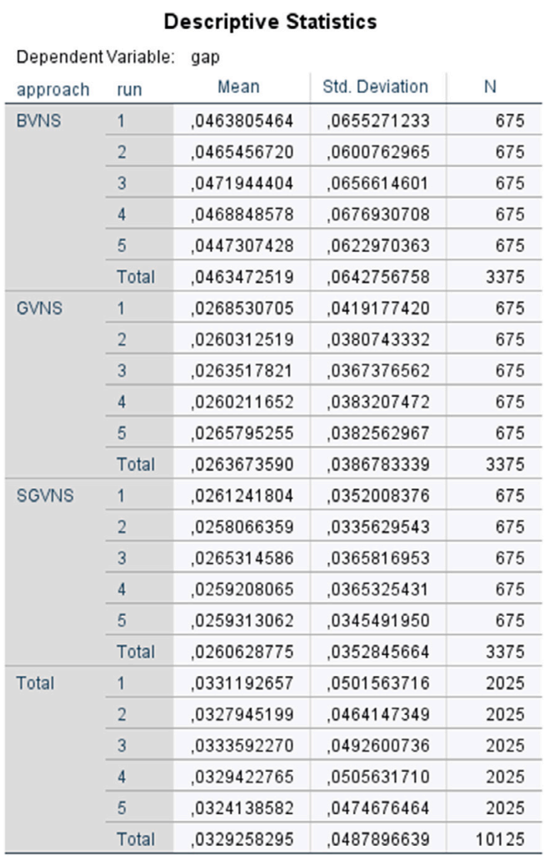
In general, all the approaches proposed in this paper are able to obtain solutions relatively close to the optimum in much less computational time than in Rietz et al. (2016).

4.4. Statistical analyses of the computational experiments

The VNS method iteratively runs through different solutions, combining stochastic methods with local search. Since the implementation embeds stochastic procedures with a random component, it is necessary to ensure that the results obtained are not biased by chance. Therefore, the solutions must achieve very similar values in each execution. Accordingly, each of the 675 instances was solved 5 times for each approach (BVNS, GVNS, and SGVNS), providing an overall of 10125 solutions (3 × 5 × 675 instances). The samples present very similar results for the 5 runs/replications in each of the proposed approaches,

Table 7
Global results for all the sets of instances (A, B and C).

set	*dim*	BVNS			GVNS			SGVNS			Exact methods (UB)		
		opt	*gap*	*t*	*opt*	*gap*	*t*	*opt*	*gap*	*t*	*opt*	*gap*	*t*
A	SI	55	0.029	1.5	63	0.017	1.6	62	0.017	1.8	68	0.230	637.0
A	LI	12	0.034	3.0	12	0.025	3.3	12	0.026	3.8	15	0.596	990.5
Global - Set A		**67**	**0.031**	**2.1**	**75**	**0.020**	**2.3**	**74**	**0.021**	**2.6**	**83**	**0.377**	**778.4**
B	SI	37	0.056	1.7	56	0.034	1.7	63	0.032	2.2	89	0.041	460.0
B	LI	0	0.060	3.7	0	0.038	4.8	0	0.039	7.0	4	0.282	805.9
Global - Set B		**37**	**0.058**	**2.7**	**56**	**0.036**	**3.2**	**63**	**0.036**	**4.6**	**93**	**0.161**	**633.0**
C	SI	111	0.001	0.6	135	0.000	0.3	135	0.000	0.5	135	0.000	0.3
C	LI	28	0.002	1.0	45	0.000	0.7	44	0.000	1.4	45	0.000	1.2
Global - Set C		**139**	**0.002**	**0.7**	**180**	**0.000**	**0.4**	**179**	**0.000**	**0.7**	**180**	**0.000**	**0.5**

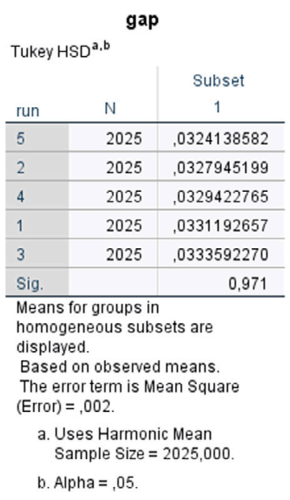


Descriptive Statistics

Dependent Variable: gap

approach	run	Mean	Std. Deviation	N
BVNS	1	,0463805464	,0655271233	675
	2	,0465456720	,0600762965	675
	3	,0471944404	,0656614601	675
	4	,0468848578	,0676930708	675
	5	,0447307428	,0622970363	675
	Total	,0463472519	,0642756758	3375
GVNS	1	,0268530705	,0419177420	675
	2	,0260312519	,0380743332	675
	3	,0263517821	,0367376562	675
	4	,0260211652	,0383207472	675
	5	,0265795255	,0382562967	675
	Total	,0263673590	,0386783339	3375
SGVNS	1	,0261241804	,0352008376	675
	2	,0258066359	,0335629543	675
	3	,0265314586	,0365816953	675
	4	,0259208065	,0365325431	675
	5	,0259313062	,0345491950	675
	Total	,0260628775	,0352845664	3375
Total	1	,0331192657	,0501563716	2025
	2	,0327945199	,0464147349	2025
	3	,0333592270	,0492600736	2025
	4	,0329422765	,0505631710	2025
	5	,0324138582	,0474676464	2025
	Total	,0329258295	,0487896639	10125

Fig. 18. Mean *gap* for each approach and each run.



gap

Tukey HSD^{a,b}

run	N	Subset
		1
5	2025	,0324138582
2	2025	,0327945199
4	2025	,0329422765
1	2025	,0331192657
3	2025	,0333592270
Sig.		0,971

Means for groups in homogeneous subsets are displayed.
Based on observed means.
The error term is Mean Square (Error) = ,002.

a. Uses Harmonic Mean Sample Size = 2025,000.

b. Alpha = ,05.

Fig. 19. Homogeneous subsets between experiments/runs.

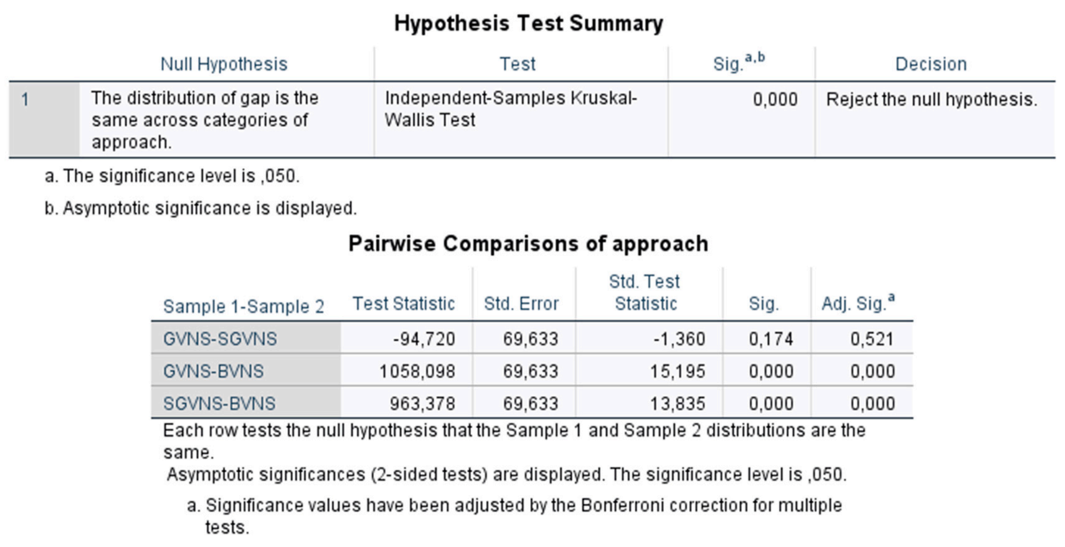
as can be verified through the analysis of the mean gap in Fig. 18 (there are no considerable differences between each run for a given approach). Moreover, a Tukey's HSD test was conducted to compare the runs using the optimality gap as the dependent variable. This test proved that there are no statistically significant differences between each run of the experiments (regardless of approach) with a significance level of 5% (Fig. 19). Thus, samples of each run showed to be homogeneous.

Since the results obtained do not follow a normal distribution (considering the optimality gap as the variable), we resort to non-parametric statistical tests in independent samples, more precisely in the Kruskal–Wallis test. By comparing the suggested VNS approaches (Fig. 20), it can be concluded that there are statistically significant differences between the BVNS and the other two approaches (GVNS and SGVNS). When comparing GVNS and SGVNS, the null hypothesis cannot be rejected: the samples are statistically identical with a significance level of 5%.

To also establish a comparison between our best solutions using the optimality gap (gap_{best}) for each of the three VNS approaches and the best results proposed by Rietz et al. (2016) (gap_{UB}), we define gap' as the difference between gap_{best} and gap_{UB} as follows: $gap' = gap_{best} - gap_{UB}$. Therefore, gap'=0 means that both approaches have the same objective function value. A negative value of gap' means that our approaches lead to results that outperform those presented by Rietz et al. (2016). In contrast, positive values of gap' indicate that our methods were not as effective.

For the sake of clarity, in Fig. 21 a set of box plots graphically demonstrates the spread of the gap'. Each box graph represents a different combination of method (BVNS, GVNS and SGVNS) and set (A, B, and C).

As can be seen, most of the box plots are in the negative zone. More precisely, in set A for all approaches, 75% of the solutions obtained are better than or equal to those presented in the literature: the third quartile (Q_3) is approximately below the line where the gap' is null. The trend in set B also shows that most solutions are closer to the optimal ones. In set C, we were also able to obtain all optimal solutions using the GVNS approach with a slightly shorter computational time than in Rietz et al. (2016).



Hypothesis Test Summary

	Null Hypothesis	Test	Sig.[a,b]	Decision
1	The distribution of gap is the same across categories of approach.	Independent-Samples Kruskal-Wallis Test	0,000	Reject the null hypothesis.

a. The significance level is ,050.

b. Asymptotic significance is displayed.

Pairwise Comparisons of approach

Sample 1-Sample 2	Test Statistic	Std. Error	Std. Test Statistic	Sig.	Adj. Sig.[a]
GVNS-SGVNS	-94,720	69,633	-1,360	0,174	0,521
GVNS-BVNS	1058,098	69,633	15,195	0,000	0,000
SGVNS-BVNS	963,378	69,633	13,835	0,000	0,000

Each row tests the null hypothesis that the Sample 1 and Sample 2 distributions are the same.
Asymptotic significances (2-sided tests) are displayed. The significance level is ,050.

a. Significance values have been adjusted by the Bonferroni correction for multiple tests.

Fig. 20. Independent-Samples Kruskal–Wallis test results.

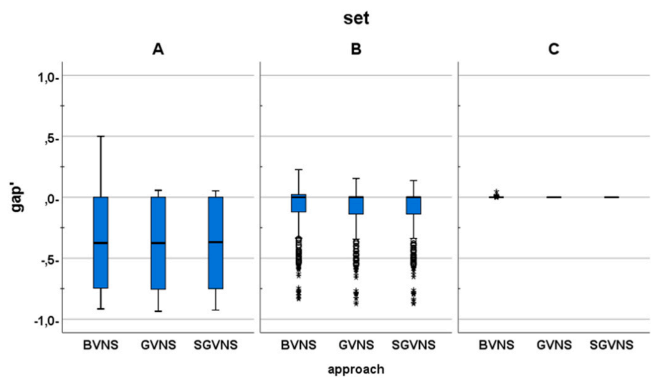


Fig. 21. Box plots of the gap' for each approach and set.

5. Conclusions

In this paper, we described and analyzed different metaheuristic algorithms for the integrated planning and scheduling problem. We proposed several variants of the variable neighborhood search algorithms, including a basic, a general and a skewed general version. New neighborhood structures were introduced and described. All the approaches were tested through extensive computational experiments using benchmark instances, and compared with the state-of-the-art methods described in the literature. Large-size benchmark instances were also solved which is still scarce in the literature.

The results obtained show that the proposed algorithms are able to obtain solutions that are very close to the optimal ones (with very small optimality gaps). One of the major advantages is the very short computing time required to reach these high-quality solutions. In general, the proposed approaches clearly outperform current state-of-the-art approaches presented so far.

CRediT authorship contribution statement

Mário Manuel Silva Leite: Conceptualization, Methodology, Software, Validation, Formal analysis, Investigation, Writing – original draft, Visualization. **Telmo Miguel Pires Pinto:** Conceptualization, Methodology, Validation, Investigation, Writing – original draft, Supervision. **Cláudio Manuel Martins Alves:** Conceptualization, Validation, Investigation, Resources, Writing – review & editing, Supervision.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

Acknowledgments

We are indebted to Lino Costa, who helped with the statistical analyses. We are also indebted to the anonymous referees for their thorough reviews. Their suggestions helped us to improve the overall quality of the paper.

Funding

The first author has been supported by FCT – Fundação para a Ciência e Tecnologia, through national funds from MCTES – Ministério da Ciência, Tecnologia e Ensino Superior, and by European Social Fund through NORTE2020 – Programa Operacional Regional Norte, within the research grant SFRH/BD/146217/2019. The second and third authors have been supported by FCT – Fundação para a Ciência e a Tecnologia within the R&D Units Project Scope: UIDB/00319/2020. The second author has been supported by national funds through FCT – Fundação para a Ciência e a Tecnologia, under the project UIDB/00285/2020.

References

Almada-Lobo, B., Oliveira, J. F., & Carravilla, M. A. (2008). Production planning and scheduling in the glass container industry: a VNS approach. *International Journal of Production Economics*, *114*(1), 363–375.

Armstrong, R., Gao, S., & Lei, L. (2008). A zero-inventory production and distribution problem with a fixed customer sequence. *Annals of Operations Research*, *159*(1), 395–414.

Bilyk, A., & Mönch, L. (2012). A variable neighborhood search approach for planning and scheduling of jobs on unrelated parallel machines. *Journal of Intelligent Manufacturing*, *23*(5), 1621–1635.

Brimberg, J., Mladenović, N., Todosijević, R., & Urošević, D. (2017). A general framework for nested variable neighborhood search. *Electronic Notes in Discrete Mathematics*, *58*, 159–166.

Cardoen, B., Demeulemeester, E., & Beliën, J. (2010). Operating room planning and scheduling: A literature review. *European Journal of Operational Research*, *201*(3), 921–932.

Cheaitou, A., & Khan, S. A. (2015). An integrated supplier selection and procurement planning model using product predesign and operational criteria. *International Journal on Interactive Design and Manufacturing*, *9*(3), 213–224.

Chen, Y. (2016). Integrated optimization model for production planning and scheduling with logistics constraints. *International Journal of Simulation Modelling*, *15*(4), 711–720.

Dogan, M. E., & Grossmann, I. (2006). A decomposition method for the simultaneous planning and scheduling of single-stage continuous multiproduct plants. *Industrial and Engineering Chemistry Research*, *45*(1), 299–315.

Erdirik-Dogan, M., & Grossmann, I. (2008). Simultaneous planning and scheduling of single-stage multi-product continuous plants with parallel lines. *Computers & Chemical Engineering*, *32*(11), 2664–2683.

Eun, J., Kim, S.-P., Yih, Y., & Tiwari, V. (2019). Scheduling elective surgery patients considering time-dependent health urgency: Modeling and solution approaches. *Omega*, *86*, 137–153.

Geismar, H. N., Laporte, G., Lei, L., & Sriskandarajah, C. (2008). The integrated production and transportation scheduling problem for a product with a short lifespan. *INFORMS Journal on Computing*, *20*(1), 21–33.

Grossmann, I. (2009). Research challenges in planning and scheduling for enterprise-wide optimization of process industries. *Computer Aided Chemical Engineering*, *27*, 15–21.

Hansen, P., Mladenović, N., Todosijević, R., & Hanafi, S. (2017). Variable neighborhood search: basics and variants. *EURO Journal on Computational Optimization*, *5*(3), 423–454.

Kis, T., & Kovács, A. (2012). A cutting plane approach for integrated planning and scheduling. *Computers & Operations Research*, *39*(2), 320–327.

Kopanos, G. M., Puigjaner, L., & Georgiadis, M. C. (2012). Simultaneous production and logistics operations planning in semicontinuous food industries. *Omega*, *40*(5), 634–650.

Liu, W.-L., Gong, Y.-J., Chen, W.-N., Liu, Z., Wang, H., & Zhang, J. (2020). Coordinated charging scheduling of electric vehicles: A mixed-variable differential evolution approach. *IEEE Transactions on Intelligent Transportation Systems*, *21*(12), 5094–5109.

Macedo, R., Alves, C., Hanafi, S., Jarboui, B., Mladenović, N., Ramos, B., & de Carvalho, J. V. (2015). Skewed general variable neighborhood search for the location routing scheduling problem. *Computers & Operations Research*, *61*, 143–152.

Maenhout, B., & Vanhoucke, M. (2013). An integrated nurse staffing and scheduling analysis for longer-term nursing staff allocation problems. *Omega*, *41*(2), 485–499.

Maravelias, C. T., & Sung, C. (2009). Integration of production planning and scheduling: Overview, challenges and opportunities. *Computers & Chemical Engineering*, *33*(12), 1919–1930.

Mladenović, N., & Hansen, P. (1997). Variable neighborhood search. *Computers & Operations Research*, *24*(11), 1097–1100.

Persson, J. A., & Göthe-Lundgren, M. (2005). Shipment planning at oil refineries using column generation and valid inequalities. *European Journal of Operational Research*, *163*(3), 631–652.

Pinto, T., Alves, C., & Valério de Carvalho, J. (2020). Variable neighborhood search algorithms for the vehicle routing problem with two-dimensional loading constraints and mixed linehauls and backhauls. *International Transactions in Operational Research*, *27*(1), 549–572.

Rietz, J., Alves, C., Braga, N., & Valério de Carvalho, J. (2016). An exact approach based on a new pseudo-polynomial network flow model for integrated planning and scheduling. *Computers & Operations Research*, *76*, 183–194.

Sel, Ç., Bilgen, B., & Bloemhof-Ruwaard, J. (2017). Planning and scheduling of the make-and-pack dairy production under lifetime uncertainty. *Applied Mathematical Modelling*, *51*, 129–144.

Shen, W., Wang, L., & Hao, Q. (2006). Agent-based distributed manufacturing process planning and scheduling: a state-of-the-art survey. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, *36*(4), 563–577.

Zhao, F., He, X., & Wang, L. (2021). A two-stage cooperative evolutionary algorithm with problem-specific knowledge for energy-efficient scheduling of no-wait flow-shop problem. *IEEE Transactions on Cybernetics*, *51*(11), 5291–5303.

Zhao, F., Zhang, L., Cao, J., & Tang, J. (2021). A cooperative water wave optimization algorithm with reinforcement learning for the distributed assembly no-idle flowshop scheduling problem. *Computers & Industrial Engineering*, *153*, Article 107082.

Zhou, S., Xing, L., Zheng, X., Du, N., Wang, L., & Zhang, Q. (2021). A self-adaptive differential evolution algorithm for scheduling a single batch-processing machine with arbitrary job sizes and release times. *IEEE Transactions on Cybernetics*, *51*(3), 1430–1442.