A Balance-First Sequence-Last Algorithm to design RMS A Matheuristic with performance guaranty to balance Reconfigurable Manufacturing Systems

Youssef Lahrichi, Laurent Deroussi, Nathalie Grangeon, Sylvie Norre

Journal of Heuristics manuscript No.
(will be inserted by the editor)

A Balance-First Sequence-Last Algorithm to design RMS

A Matheuristic with performance guaranty to balance Reconfigurable Manufacturing Systems

Youssef Lahrichi · Laurent Deroussi · Nathalie Grangeon · Sylvie Norre

Received: date / Accepted: date

Abstract The Reconfigurable Transfer Line Balancing Problem (RTLB) is considered in this paper. This problem is quite recent and motivated by the growing need of reconfigurability in the new industry 4.0 context. The problem consists into allocating a set of operations necessary to machine a single part to different workstations placed into a serial line. Each workstation can contain multiple machines operating in parallel and the tasks allocated to a workstation should be sequenced since sequence-dependent setup times between operations are needed to perform tool changes. Besides, precedence constraints, inclusion, exclusion and accessibility constraints between operations are considered. In this article we propose an efficient matheuristic of type BFSL (Balance First, Sequence Last). This method is a two-step heuristic with a constructive phase and an improvement phase. It contains several components from exact methods (linear programming, constraint generation and dynamic

Y. Lahrichi
LIMOS CNRS UMR 6158, 1 Rue de la Chebarde, 63178 Aubière, FRANCE
Tel.: +33-473-407580
E-mail: youssef.lahrichi.contact@gmail.com

L. Deroussi
IUT d'Allier, LIMOS CNRS UMR 6158, Avenue Aristide Briand, 03100 Montluçon, FRANCE
Tel.: +33-470-022082
E-mail: laurent.deroussi@uca.fr

N. Grangeon
IUT d'Allier, LIMOS CNRS UMR 6158, Avenue Aristide Briand, 03100 Montluçon, FRANCE
Tel.: +33-470-022082
E-mail: nathalie.grangeon@uca.fr

S. Norre
IUT d'Allier, LIMOS CNRS UMR 6158, Avenue Aristide Briand, 03100 Montluçon, FRANCE
Tel.: +33-470-022099
E-mail: sylvie.norre@uca.fr

programming) and metaheuristics (simulated annealing). In addition, we show that the constructive algorithm approximates the optimal solution when the setup times are bounded by the processing times and give a approximation ratio. The obtained results show the effectiveness of the proposed approach. The matheuristic clearly outperforms a genetic algorithm from literature on quite large benchmark instances.

Keywords RTLB · Line balancing · Constraint generation · Simulated annealing · matheuristic · Approximation algorithm

1 Introduction

Nowadays, manufacturers are faced with a highly volatile market bringing a growing variety in demand. The modern manufacturing system should be able to be reconfigured to adapt to the new market demand within low cost and time. Two issues must be tackled: the variability in production size and the variability in the product specifications. While the first one imposes a variation in cycle time (period of time that separates the exit of two consecutive processed products from the production line), the second one is linked to the set of tasks involved in the production process.

To address this issue, Koren [8] suggested in late 1990s the novel concept of Reconfigurable Manufacturing System (RMS). The RMS we consider in this paper is a serial line of workstations, each workstation being equipped by multiple machines operating in parallel (Fig. 1 represents a RMS with 3 workstations, the first workstation is equipped by 3 machines, the second one with 2 machines and the third one with 3 machines). Part units are moved from a workstation to another thanks to a conveyor. The part is delivered then to the first available machine in a workstation by the gantry.

The RMS can be reconfigured to accommodate a change in production size. Indeed, the cycle time can be monitored by adding or removing a machine in a workstation. This property is known as scalability [10]. Besides, RMS offers a good tradeoff between productivity and flexibility while *Dedicated Manufacturing System* are highly productive but very poorly flexible and *Flexible Manufacturing System* highly flexible but very expensive. ([8],[9])

Despite being scalable and profitable, RMS could not address the issue of variability in product specifications unless they are equipped with machines that can handle a large set of operations. Mono-spindle head machines can perform a big set of operations, each machine being equipped with a tool magazine. A specific tool is needed to perform a specific operation. Thus, we consider sequence-dependent setup times between operations in order to account for tool changes in addition to processing times of operations.

Once equipped with mono-spindle head machines, RMS addresses both the production size and product specifications variability issues: whenever one or both of these elements come to change, the manufacturer can easily adjust the production by performing a *reconfiguration* of the system which can be seen as a line balancing problem with sequence-dependent setup times and

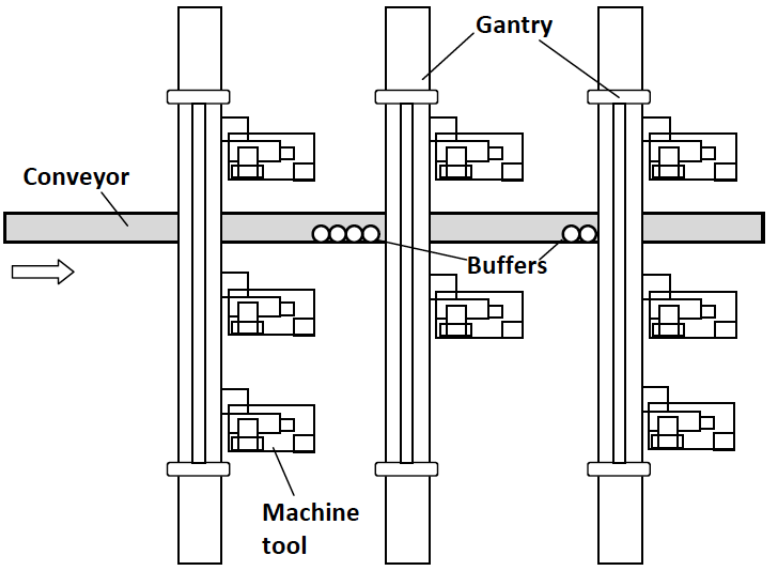

Fig. 1: Reconfigurable Manufacturing System

multiple parallel machines at each workstation (also called parallel workstation). Machines are then added or removed from workstations if necessary, and machines are remotely configured to perform the new sequence of operations. RMS allows to perform those two steps rapidly and cost-effectively.

The problem dealt with in this paper is called the *Reconfigurable Transfer Line Balancing Problem (RTLB)*. It considers constraints coming from the machining industry. We deal with the problem of balancing the line for a single product. The remainder of the paper is organized as follows: Sect. 2 defines the problem with a small numerical example and an ILP formulation, Sect. 3 gives the related work on the problem. Sect. 4 gives the general scheme of the matheuristic, Sect. 5 introduces the construction method and Sect. 6 introduces the improvement method. An experimental study is conducted in Sect. 7.

2 Presentation of the problem

2.1 Problem definition

We are given a set of operations necessary to machine a part (product). The production line considered is a serial line of workstations. Each workstation is equipped with multiple machines working in parallel. Each product goes through all the workstations of the line in the order. Each product is processed by only one machine at each station. At each cycle time, a new product

arrives on the line on the first workstation, a processed product leaves the line from the last workstation and in each workstation a product leaves to the next workstation and is replaced by a product coming from the precedent workstation. The machines within the same workstation perform the same sequence of operations and machines from different workstations perform different sequences of operations.

The optimization problem considered consists in:

- Assigning the operations to the workstations (Balancing subproblem).
- Sequencing the operations in each workstation (Sequencing subproblem).
- Determining the number of machines in each workstation.

The objective considered is to minimize the overall number of machines used such that the following constraints are respected:

- The number of workstations must not exceed the maximum number of workstations.
- The number of operations allocated to a workstation must not exceed the maximum number of operations per workstation.
- The number of machines in a workstation must not exceed the maximum number of machines per workstation.
- Precedence constraints must be respected: when an operation i precedes an operation j, either the workstation to which the operation i is allocated must be before the workstation to which the operation j is allocated or i and j are assigned to the same workstation and i is processed before j.
- For each workstation, the workload (the sum of the processing times and the setup times induced by the sequence allocated to the workstation) divided by the number of machines allocated to the workstation must not exceed the cycle time.
- Inclusion constraints must be respected: each constraint links two operations that must be assigned to the same workstation.
- Exclusion constraints must be respected: they consist in subsets of operations (called exclusion sets) such that all the operations belonging to the same subset cannot be assigned to the same workstation. But the operations of any proper subset of an exclusion set are allowed to be assigned to the same workstation. We denote by E the set of all exclusion subsets. For example, suppose we have 4 operations denoted o_1, o_2, o_3, o_4 and an exclusion set $\{o_1, o_2, o_3\}$. In this case, it is not acceptable to have o_1, o_2 and o_3 all assigned to the same workstation. However, it is possible to have o_1 and o_2 assigned to the same workstation provided o_3 is assigned to a different workstation.
- Accessibility constraints must be respected: each operation i has a subset Pos_i of possible part-fixing positions. An accessibility constraint is related to a workstation. It imposes that all the operations assigned to the same workstation must have at least one common part-fixing position. For example, suppose we have 4 operations denoted o_1, o_2, o_3, o_4 and 3 possible part-fixing positions $Pos = \{1, 2, 3\}$ such that:

$$Pos_{o_1} = \{1, 2\}, Pos_{o_2} = \{1, 2, 3\}, Pos_{o_3} = \{2, 3\}, Pos_{o_4} = \{3\}$$

Operations $\{o_1, o_2, o_3\}$ could be assigned to the same workstation because the position 2 is shared by o_1, o_2 and o_3. However operations $\{o_1, o_2, o_4\}$ could not be assigned to the same workstation because they share no position.

The inclusion, exclusion and accessibility constraints come from the machining industry.

For the rest of the paper, we use the notations presented in Table 1.

Table 1: Table of notations.

n	Number of operations
s_{max}	Maximum number of workstations
N	Set of operations, indexed on $\{1, 2, \ldots, n\}$
S	Set of workstations, indexed on $\{1, 2, \ldots, s_{max}\}$
P	Set of couples $(i, j) \in N \times N$ such that i precedes j (also denoted $i << j$)
M_{max}	Maximum number of machines in a workstation
N_{max}	Maximum number of operations in a workstation
C	Cycle time.
d_i	Processing time of operation i.
$t_{i,j}$	Setup time to be considered when operation i is performed just before operation j in some workstation
I	Set of couples $(i, j) \in N \times N$ linked with an inclusion constraint
E	Set of subsets of operations that cannot be assigned to the same workstation
Pos	Set of all possible part-fixing positions.
Pos_i	Subset of possible part-fixing positions for operation i.

2.2 Example

We illustrate the studied problem with a small instance described by the following data:

- The part requires the execution of 6 operations numbered from 1 to 6 ($n = 6$).
- Precedence constraints are described by the following:

$$P = \{(1,3), (3,4), (3,5), (2,3), (4,6), (5,6)\}$$

represented by Fig. 2.

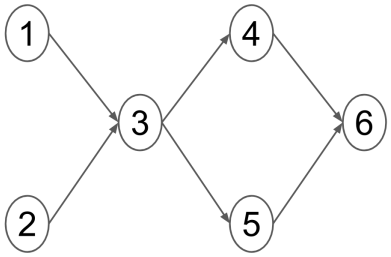

Fig. 2: Precedence graph.

– At most 3 workstations can be used: $s_{max} = 3$.
– Processing times are represented in Table 2.

Table 2: Processing times.

i	1	2	3	4	5	6
d_i	1.5	1	3.5	1.5	2.5	1.5

– Setup times are represented in Table 3.

Table 3: Setup times.

$t_{i,j}$	$j = 1$	2	3	4	5	6
$i = 1$	0	0.5	1	1	1	
2	1	0	0.5	1	1	1
3	0.5	1	0	1	1	1
4	1	1	1	0	0.5	0.5
5	1	1	1	0.5	0	1
6	1	1	1	1	0.5	0

– Maximum number of operations that could be assigned to a workstation: $N_{max} = 3$.
– Maximum number of machines that could be hosted by a workstation: $M_{max} = 3$.
– C = 2.5, cycle time.
– Inclusion and exclusion constraints are given by:

$$I = \{(1, 2)\}, E = \{\{3, 4, 5\}\}$$

– Accessibility constraints are given as follows: $Pos = \{1, 2, 3, 4\}$, $Pos_4 = \{1, 2\}$, $Pos_5 = \{1, 4\}$, $Pos_i = \{1, 2, 3, 4\}, \forall i \in \{1, 2, 3, 6\}$

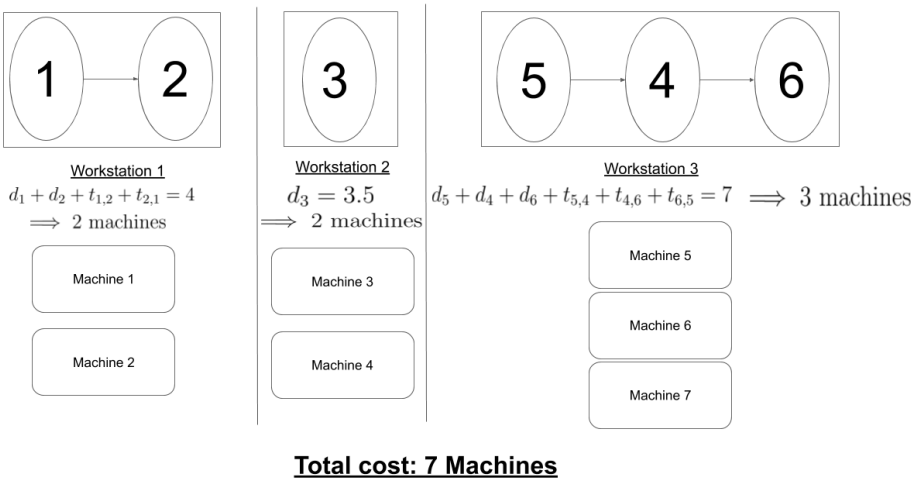

Fig. 3: Example of feasible solution.

A feasible solution is represented in Fig. 3. $\lceil \frac{d_1+d_2+t_{1,2}+t_{2,1}}{C} \rceil = \lceil \frac{4}{2.5} \rceil = 2$, so 2 machines are required for workstation 1. The machines that must be hosted by the other workstations are indicated in Fig. 3. In this example, the total number of machines required to process all the operations is 7

2.3 Mathematical formulation

In order to clarify the definition of the problem, we describe the ILP proposed for the RTLB in [11]. The approach is based on modelling the sequences of operations assigned to each workstation. It uses the following binary variables:

$$x_{i,s,j} = \begin{cases} 1 \text{ If operation } i \text{ is assigned to workstation } s \text{ at the } j^{th} \text{ position of its sequence.} \\ 0 \text{ Otherwise.} \end{cases}$$

$$y_s = \begin{cases} 1 \text{ If at least one operation is assigned to workstation } s \\ 0 \text{ Otherwise.} \end{cases}$$

$$v_{s,k} = \begin{cases} 1 \text{ If } k \text{ machines are assigned to workstation } s. \\ 0 \text{ Otherwise.} \end{cases}$$

$$z_{i,i',s} = \begin{cases} 1 \text{ If operation } i \text{ is processed just before operation } i' \text{ at workstation } s. \\ 0 \text{ Otherwise.} \end{cases}$$

$$w_{i,s} = \begin{cases} 1 \text{ If operation } i \text{ is assigned to the last position of the sequence at workstation } s. \\ 0 \text{ Otherwise.} \end{cases}$$

$$u_{s,a} = \begin{cases} 1 \text{ If part-fixing position } a \text{ is chosen for workstation } s. \\ 0 \text{ Otherwise.} \end{cases}$$

We consider the objective of minimizing the number of machines used:

$$Min \sum_{s=1}^{s_{max}} \sum_{k=1}^{M_{max}} k.v_{s,k} \tag{1}$$

under the constraints: (2-16).

$$\sum_{s=1}^{s_{max}} \sum_{j=1}^{N_{max}} x_{i,s,j} = 1, \forall i \in N \tag{2}$$

$$\sum_{i=1}^{n} x_{i,s,j} \leq 1, \forall s \in S, \forall j \in \{1, \ldots, N_{max}\} \tag{3}$$

$$\sum_{i=1}^{n} x_{i,s,j+1} \leq \sum_{i=1}^{n} x_{i,s,j}, \forall s \in S, \forall j \in \{1, \ldots, N_{max} - 1\} \tag{4}$$

$$\sum_{k=1}^{M_{max}} v_{s,k} = y_s, \forall s \in S \tag{5}$$

$$y_{s+1} \leq y_s, \forall s \in \{1, \ldots, s_{max} - 1\} \tag{6}$$

$$0 \leq \sum_{s=1}^{s_{max}} \sum_{j=1}^{N_{max}} (N_{max}.(s-1) + j)(x_{i',s,j} - x_{i,s,j}), \forall (i, i') \in P \tag{7}$$

$$\sum_{i=1}^{n} \sum_{j=1}^{N_{max}} d_i.x_{i,s,j} + \sum_{i=1}^{n} \sum_{i'=1}^{n} t_{i,i'}.z_{i,i',s} \leq C. \sum_{k=1}^{M_{max}} k.v_{s,k}, \forall s \in S \tag{8}$$

$$x_{i,s,j} + x_{i',s,j+1} \leq 1 + z_{i,i',s}, \forall (i, i') \in N^2, i \neq i', 0 \leq j < N_{max}, \forall s \in S \tag{9}$$

$$x_{i,s,j} - \sum_{i' \in N; i' \neq i} x_{i',s,j+1} \leq w_{i,s}, \forall i \in N, \forall s \in S, \forall j \in \{1, \ldots, N_{max} - 1\} \tag{10}$$

$$x_{i,s,N_{max}} \leq w_{i,s}, \forall i \in N, \forall s \in S \tag{11}$$

$$w_{i,s} + x_{i',s,1} \leq 1 + z_{i,i',s}, \forall (i, i') \in N^2, i \neq i', \forall s \in S \tag{12}$$

$$\sum_{s=1}^{s_{max}} \sum_{j=1}^{N_{max}} s.x_{i,s,j} = \sum_{s=1}^{s_{max}} \sum_{j=1}^{N_{max}} s.x_{i',s,j}, \forall (i, i') \in I \tag{13}$$

$$\sum_{i \in ES} \sum_{j=1}^{N_{max}} x_{i,s,j} \leq |ES| - 1, \forall ES \in E, \forall s \in S \tag{14}$$

$$\sum_{a \in Pos} u_{s,a} \leq 1, \forall s \in S \tag{15}$$

$$\sum_{j=1}^{N_{max}} x_{i,s,j} - \sum_{a \in Pos_i} u_{s,a} \leq 0, \forall i \in N, \forall s \in S \tag{16}$$

The set of constraints (2) ensures that each operation is assigned to exactly one workstation at a unique position of its sequence. (3) ensures that at each workstation at most one operation is assigned in each position of the sequence. (4) ensures that at each workstation no operation is assigned at a position $j+1$ unless some operation is assigned at the position j. (5) ensures that only one number of machines is chosen for every used workstation. (6) ensures that no workstation is used unless its precedent workstation is also used. (7) ensures that precedence constraints are satisfied. (8) ensures that the cycle time is not exceeded at any workstation. (9) ensures that if operation i is followed by operation i' at workstation s then $z_{i,i',s}$ is set to 1. Constraints (10) and (11) ensure that $w_{i,s}$ is set to 1 whenever operation i is positioned at the last occupied position in the sequence of workstation s. (12) ensures that if operation i is positioned at the last occupied position in the sequence of workstation s and operation i' is positioned at the first position in the sequence of workstation s then $z_{i,i',s} = 1$ and consequently the setup time $t_{i,i'}$ is considered in (8). (13) ensures that inclusion constraints are satisfied while (14) insures that exclusion constraints are satisfied. (15) and (16) ensure that accessibility constraints are satisfied.

3 Related work

Academically, the RTLB problem could be seen as an assembly line balancing problem. Those problems have been well studied in the literature [1]; however those considering parallel machines or sequence dependent setup times have rarely been considered. The originality of the problem comes from the consideration of both elements.

The problem is a generalisation of the Simple Assembly Line Balancing problem which is known to be NP-Hard [13]. Few papers deal with the RTLB problem, it has been first introduced in [4] where a MIP approach was introduced. Another ILP was lately suggested in [11].

An exact method was suggested in [2]: it uses a set partitioning model coupled with a constraint generation algorithm.

In [6], an ant colony optimization algorithm was suggested for the problem. Every ant creates a workstation and adds operations to it as long as it is possible. The ants should decide whether to put an operation over another in the current workstation. This is done thanks to probability distribution given by pheromone trails.

In [3], a greedy construction algorithm is suggested to give a solution. There is no guaranty however that the algorithm will give a feasible solution. Its idea is to try to add operations from a list of candidate operations to the current workstation as long as it is possible. Once the workstation is full, another workstation is created, and the process is repeated until all operations are assigned.

Since the greedy algorithm could output a solution that exceeds the number of workstations authorized, a second phase is developed to merge workstations if possible.

In [5], a GRASP approach is introduced. Starting solutions are generated using the greedy random heuristic of [3] then a local search is performed using 4 neighbourhoods:

- Move a subset of operations from a workstation to another.
- Merge one or more workstations.
- Move operations from most loaded workstations to less loaded workstations.
- Optimize the subsequence of operations in each workstation (using heuristics and a MIP).

In [2], a genetic algorithm is suggested. A chromosome is coded as a giant sequence of all operations and a heuristic decoder is suggested to build a solution. Classical crossover operators for the TSP are used as well as the following mutation operators:

- Swap 2 operations.
- Insert an operation into a different position.

Unfortunately, exact methods of the literature fail to solve instances of industrial size. Besides, no resolution method guaranties the feasibility of the solution for large scale instances. To solve the problem, 2 subproblems must be solved jointly: the balancing subproblem and the sequencing subproblem. Therefore, resolution methods for the problem can be categorized within 3 classes:

- BFSL (Balance-First, Sequence-Last): The balancing subproblem is solved first then the sequencing subproblem.
- SFBL (Sequence-First, Balance-Last): The sequencing subproblem is solved first then the balancing subproblem.
- Integrated approaches: The balancing subproblem and the sequencing subproblem are solved in the same time.

This paper investigates a sequential approach of type BFSL.

4 The matheuristic general scheme

The proposed matheuristic (MATH-BFSL) is a two-phase heuristic: a construction phase (I-BFSL: Iterated BFSL) and an improvement phase. The general scheme of this method is given in Fig. 4.

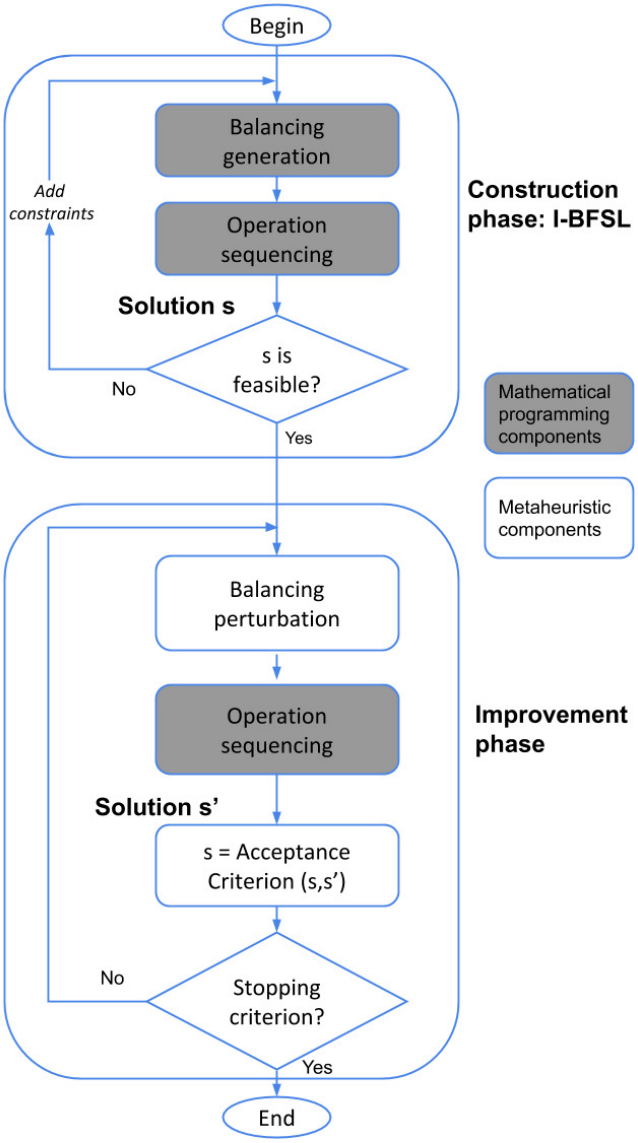

Fig. 4: General scheme of the proposed matheuristic

Within each of these phases, a solution is generated, that is denoted by s for the construction phase, and by s' for the improvement phase. The same idea is applied for the generation of s and s'. First, we solve the balancing subproblem, which consists in allocating the operations to the workstations. Then, for each workstation, the assigned operations must be sequenced in order

to minimize the sum of the setup times, and therefore the number of machines required to satisfy the cycle time constraint.

An acceptance criterion is defined at the end of the improvement phase, in order to choose which of the two solutions will be kept for the next iteration. Moreover, we indicate in Fig. 4 which components come from the field of mathematical programming, and which ones come from the field of metaheuristics.

5 Construction phase: I-BFSL

The construction phase is a two-step iterative method piloted by a constraint generation algorithm:

- Balancing generation: The first step consists in solving the line balancing problem without consideration of the sequence-dependent setup times. In this case, the decisions to make are which operations to assign to which workstation and how much machines are needed in each workstation. We solve this problem thanks to an ILP.
- Operations sequencing: In the second step, we consider sequence-dependent setup times. This implies the decision of sequencing the operations in each workstation taking as input the solution of the first step. The sequencing problem is optimally solved thanks to a dynamic programming algorithm.
- Constraint generation: Due to the consideration of sequence-dependent setup times in the second step, the load of the workstations increases. This can result in exceeding the maximum number of machines authorized in some workstation. For this reason, constraints are iteratively generated and added to the model of the first step. Those constraints aim at forbidding solutions where the maximum number of machines authorized in the workstations is exceeded.

Each of the 3 previous points are explained separately in the following 3 subsections.

5.1 Balancing generation

The model of the RTLB, described in Sect. 2, cannot be used to solve medium to large size instances. We propose to use a relaxed version in which the sequence of operations in each workstations is not take into account. As a consequence, variables $x_{i,s,j}$ of the RTLB model becomes $x_{i,s}$ since we don't need to deal with the position in the sequence. Moreover, the variables $z_{i,i',s}$ and $w_{i,s}$ are no longer useful. All the constraints that use these variables are modified and we propose a simplified model described below. Because the setup times are not considered, the total completion time computed for each workstation may be significantly inferior to the real value. In order to give a better lower

bound, we consider for each operation i a setup time t_i^{min} equal to the smallest setup time to or from operation i:

$$t_i^{min} = Min(Min_{j \in N, j \neq i}\{t_{i,j}\}, Min_{j \in N, j \neq i}\{t_{j,i}\})$$

And so a better lower bound for workstation s is given for the total completion time by the formula:

- If at least 2 operations are assigned to workstation s:

$$\sum_{i \in N} (d_i + t_i^{min}).x_{i,s}$$

- If only one operation is assigned to workstation s:

$$\sum_{i \in N} d_i.x_{i,s}$$

A setup time t_i^{min} is considered for an operation i in a workstation s only if operation i is assigned to s and if another operation different from i is also assigned to s. For this reason, we consider a variable $\tilde{x}_{i,s}$ to decide whether an operation i is assigned to workstation s with another operation.

We use the following decision variables:

$$x_{i,s} = \begin{cases} 1 \text{ If operation } i \text{ is assigned to workstation } s. \\ 0 \text{ Otherwise.} \end{cases}$$

$$y_s = \begin{cases} 1 \text{ If at least one operation is assigned to workstation } s \\ 0 \text{ Otherwise.} \end{cases}$$

$$v_{s,k} = \begin{cases} 1 \text{ If } k \text{ machines are assigned to workstation } s. \\ 0 \text{ Otherwise.} \end{cases}$$

$$u_{s,a} = \begin{cases} 1 \text{ If position } a \text{ is chosen for workstation } s. \\ 0 \text{ Otherwise.} \end{cases}$$

$$\tilde{w}_s = \begin{cases} 1 \text{ If at least 2 operations are assigned to workstation } s. \\ 0 \text{ Otherwise.} \end{cases}$$

$$\tilde{x}_{i,s} = \begin{cases} 1 \text{ If operation } i \text{ is assigned to workstation } s \text{ and } i \text{ is not the only} \\ \quad \text{operation assigned to workstation } s. \\ 0 \text{ Otherwise.} \end{cases}$$

We consider the objective of minimizing the number of machines used:

$$Min \sum_{s=1}^{s_{max}} \sum_{k=1}^{M_{max}} k.v_{s,k} \tag{17}$$

under the constraints:

$$\sum_{s=1}^{s_{max}} x_{i,s} = 1, \forall i \in N \tag{18}$$

$$\sum_{k=1}^{M_{max}} v_{s,k} = y_s, \forall s \in S \tag{19}$$

$$\sum_{i=1}^{n} x_{i,s} \leq y_s.N_{max}, \forall s \in S \tag{20}$$

$$y_{s+1} \leq y_s, \forall s \in S - \{s_{max}\} \tag{21}$$

$$\sum_{s=1}^{s_{max}} s.x_{i,s} \leq \sum_{s=1}^{s_{max}} s.x_{i',s}, \forall (i,i') \in P \tag{22}$$

$$2.\tilde{w}_s \leq \sum_{i=1}^{n} x_{i,s}, \forall s \in S \tag{23}$$

$$\sum_{i=1}^{n} x_{i,s} - 1 \leq \tilde{w}_s.N_{max}, \forall s \in S \tag{24}$$

$$x_{i,s} + \tilde{w}_s \leq \tilde{x}_{i,s} + 1, \forall i \in N, \forall s \in S \tag{25}$$

$$\tilde{x}_{i,s} \leq \sum_{i'=1, i \neq i'}^{n} x_{i',s}, \forall i \in N, \forall s \in S \tag{26}$$

$$\tilde{x}_{i,s} \leq x_{i,s}, \forall i \in N, \forall s \in S \tag{27}$$

$$\sum_{i=1}^{n} (d_i.x_{i,s} + t_i^{min}.\tilde{x}_{i,s}) \leq C. \sum_{k=1}^{M_{max}} k.v_{s,k}, \forall s \in S \tag{28}$$

$$\sum_{s \in S} s.x_{i,s} = \sum_{s \in S} s.x_{i',s}, \forall (i,i') \in I \tag{29}$$

$$\sum_{i \in ES} x_{i,s} \leq |ES| - 1, \forall ES \in E, \forall s \in S \tag{30}$$

$$\sum_{a \in Pos} u_{s,a} \leq 1, \forall s \in S \tag{31}$$

$$x_{i,s} - \sum_{a \in Pos_i} u_{s,a} \leq 0, \forall i \in N, \forall s \in S \tag{32}$$

(18) ensures that each operation is assigned to exactly one workstation. (19) ensures that only one number of machines is chosen for every used workstation. (20) ensures that the maximum number of operations to be allocated to a workstation is respected. (21) ensures that no workstation is used unless its precedent workstation is also used. (22) ensures that precedence constraints are satisfied. (23) and (24) ensure that $\tilde{w}_s = 1$ if and only if at least 2 operations are assigned to workstation s. (25)-(27) ensures that $\tilde{x}_{i,s} = 1$ if and only if at least 2 operations including i are assigned to workstation s. (28) ensures that the cycle time is not exceeded in any workstation. (29) ensures that inclusion constraints are satisfied. (30) ensures that exclusion constraints are satisfied. (31) and (32) ensure that accessibility constraints are satisfied.

5.2 Operations sequencing

From balancing generation, we are given the assignment of operations to the workstations, we are now concerned with sequencing the operations in every workstation.

The sequencing problem is an ATSP (*Asymmetric Travelling Salesman Problem*) where operations represent cities and setup times distances between cities. However, we must consider precedence constraints within the same workstations, the problem induced is an ATSP with precedence constraints sometimes referred to in the literature as the *Precedence Constrained Travelling Salesman Problem* (PCTSP) as stated in [12].

To solve this problem we consider the well-known Held and Karp algorithm [7]. The resulting algorithm is of complexity within $O(m^2.2^m)$ where m is the number of cities. We apply this algorithm to every workstation. The number of operations in each workstation is limited to N_{max} therefore the complexity is within $O(N_{max}^2.2^{N_{max}})$. Even for large industrial instances N_{max} rarely exceeds 15 which make the approach reasonable to use (for $N_{max} = 15$ we have $N_{max}^2.2^{N_{max}} = 7,372,800$).

We suppose we are placed in some workstation to which m operations are assigned. We number the operations from 1 to m. We want to construct a tour starting from operation 1 and returning to 1 passing by all the operations such that precedence constraints are respected and such that the total sum of setup times over the tour is minimized. We put $T = \{1, \ldots, m\}$. Given a subset $U \subset T$ such that $1 \in T$ and an operation i such that $i \in U$ and $i > 1$, the idea is to use the following dynamic programming formula:

$$c(U, i) = Min_{j \in U - \{i\}}(c(U - \{i\}, j) + t_{j,i}) \tag{33}$$

where $c(U, i)$ is the minimum cost of a path going from 1 to i and passing by each operation of $U - \{1, i\}$ exactly once. We recall that $t_{j,i}$ is the setup time from operation j to operation i. The dynamic programming formula is initialized as follows:

$$c(U, 1) = +\infty; \forall U \subset T \text{ such that } 1 \in U \text{ and } |U| \geq 2 \tag{34}$$

$$c(\{1\}, 1) = 0 \tag{35}$$

To take precedence constraints into consideration, we must put:

$$c(U, i) = +\infty; \forall (U, i) \text{ such that } i \in U \text{ and } i \text{ has a successor in } U \tag{36}$$

(33)-(36) give the dynamic programming formula in order to compute all the $c(U, i), \forall (U, i)$ such that $i \in U$. We do not use a recursive function to avoid the computation of the same $c(U, i)$ several times. We compute $C(U, i), i \in U$ starting from the sets U such that $|U| = 2$ to $|U| = m$ (increasing cardinality). We then compute the cost of the optimal solution as follows:

$$c^* = Min_{j \in N - \{1\}}(c(N, j) + d_{j,1}) \tag{37}$$

This guaranties the optimality and the feasibility of the solution. The algorithm is depicted in Algorithm 1.

Algorithm 1 Dynamic programming sequencing algorithm

INPUT: An instance of the RTLB problem. A subset T of m operations numbered from 1 to m.
OUTPUT: A sequence of the operations of T minimizing the sum of the setup times.

$c(\{1\}, 1) = 0$
for $k = 2$ to n **do**
  for $U \subset T$ such that $|U| = k$ and $1 \in U$ **do**
    $c(U, 1) = +\infty$
    for $i \in U$ such that $i > 1$ **do**
      if i has a successor in S **then**
        $c(U, i) = +\infty$
      else
        $c(U, i) = Min_{j \in S - \{i\}}(c(U - \{i\}, j) + d_{j,i})$
      end if
    end for
  end for
end for
$c^* = Min_{j \in T - \{1\}}(c(T, j) + d_{j,1})$
Decode the solution of cost c^* to build an optimal sequence.

5.3 Constraint generation

We denote by $\mathcal{ILP}(\mathcal{C})$ the ILP mentioned in balancing generation (Subsect. 5.1) under the set of constraints $\mathcal{C}$ (initially $\mathcal{C} = (18) - (32)$).

After solving the ILP from balancing generation, the sequence-dependent setup times are taken into consideration by running the dynamic programming algorithm from operations sequencing in each workstation. Since sequence-dependent setup times were not taken into consideration in balancing generation, we can end up with a solution exceeding the maximum number of machines allowed in some workstation after operations sequencing. To tackle this issue, constraints are iteratively added to $\mathcal{C}$. Those constraints aim at forbidding the assignment of the set of operations leading to the violation of the M_{max} constraint to the same workstation. More formally, let us denote by *sol* the solution obtained after applying balancing generation and operations sequencing. If *sol* is feasible then the construction phase is finished and the improvement phase starts. If *sol* violates the M_{max} constraint in a workstation where the set of operations $\tilde{U}$ is assigned then we iterate throw balancing generation by adding the following constraints to the ILP:

$$\sum_{u \in \tilde{U}} x_{u,s} - \sum_{u \in N - \tilde{U}} x_{u,s} \leq |\tilde{U}| - 1, \forall s \in S \qquad (C_{\tilde{U}})$$

All in all, I-BFSL is a constraint generation algorithm. It is depicted in Algorithm 2. In the beginning only constraints (18)-(32) are considered. If we

end up with an unfeasible solution, then constraints $C_{\tilde{U}}$ cutting the unfeasible solution are iteratively generated:

Algorithm 2 I-BFSL

INPUT: An instance of RTLB.
OUTPUT: s : A feasible solution.

1: Initialize the set of constraints: $\mathcal{C} := (18) - (32)$
2: Solve the ILP of balancing generation: $s = solve[\mathcal{ILP}(\mathcal{C})]$
3: Run Algorithm 1 of operations sequencing in each workstation (s is modified)
4: **while** s is not feasible **do**
5: **for all** $\tilde{U}$ set of operations assigned to a workstation where the M_{max} constraint is violated **do**
6: $\mathcal{C} = \mathcal{C} \cup C_{\tilde{U}}$
7: **end for**
8: Solve the ILP of balancing generation: $s = solve[\mathcal{ILP}(\mathcal{C})]$
9: Run Algorithm 1 of operations sequencing in each workstation (s is modified)
10: **end while**

Theorem 1 *I-BFSL terminates and outputs a feasible solution within a finite number of iterations.*

Proof In the worst case, I-BFSL forbids all sets $\tilde{U}$ that leads to exceeding the maximum number of machines M_{max}. It has then at most as many iterations as the number of sets $\tilde{U}$ it forbids plus one (this number is bounded by the number of combinations of at most N_{max} elements among n). The solution outputted is feasible since all the constraints except M_{max} constraint were taken into consideration in the balancing generation step and the M_{max} constraint is considered in the constraint generation.

Theorem 2 *I-BFSL is a 2-approximation algorithm if we assume that:*

$$t_{i,j} \leq min(d_i, d_j) \quad \forall (i,j) \in N^2 \tag{38}$$

Proof The solution outputted by the algorithm is feasible and its overall cost (denoted c) is given by the cost of the solution outputted by the balancing generation (denoted c_1) plus the number of machines added following operations sequencing due to the consideration of setup times (denoted m), i.e $c = c_1 + m$. Besides we have: $c_1 \leq c^*$ where c^* denotes the optimal solution of the RTLB balancing problem. Thanks to (38) we have $m \leq c_1$ because the number of setup times for each workstation is less or equal to the number of operations. Then the workload induced by the setup times in each workstation is less or equal to the workload induced by the operations times. Those two inequations ($c_1 \leq c^*$, $m \leq c_1$) finally give:

$$c \leq 2.c^*$$

which shows the approximation ratio.

Even if both subproblems (balancing and sequencing) are solved to optimality in I-BFSL, the given solution is not guaranteed to be optimal because the two subproblems are not solved jointly. For this reason, an improving method is suggested after I-BFSL.

6 Improvement phase

Once we have computed a solution with I-BFSL, we suggest to improve it by means of an adaptive simulated annealing [14]. We first describe the encoding scheme of a solution in Subsect. 6.1, then the neighbourhood systems in Subsect. 6.2 and the scheme of the metaheuristic in Subsect. 6.3.

6.1 Solution encoding

A solution is encoded as a list of workstations. Each workstation is encoded by the sequence of operations assigned to it. More formally, a solution s with p workstations is represented as a $p-$tuple:

$$s = (s_1, s_2, \ldots, s_p)$$

where s_k is the sequence of operations allocated to workstation k. If q operations are allocated to workstation k, then s_k is represented as follows:

$$s_k = (s_k^1, s_k^2, \ldots, s_k^q)$$

where s_k^i is the i^{th} operation allocated to workstation k.

6.2 Neighbourhood systems

We use 3 neighbourhood operators:

- V_1 Insertion (of an operation from a workstation to another workstation).
- V_2 Merger (of two workstations).
- V_3 Split (of one workstation to two workstations).

The insertion, merger and split moves are described in (respectively) Algorithm 3, Algorithm 4 and Algorithm 5.

We notice that the moves are performed in such way that:

- For V_1, V_2 and V_3: precedence and inclusion constraints are respected.
- For V_3: exclusion constraints are respected.

If the obtained solution does not respect the other constraints then a new solution is generated (Algorithm 6).

Algorithm 3 Principle algorithm for insertion move

INPUT: s A feasible solution of the RTLB problem.
OUTPUT: s' A random neighbour of s with respect to insertion move

1: Select randomly and uniformly an operation i (the workstation of i is called the source workstation).
2: Select randomly and uniformly a destination workstation. This workstation is chosen between the last workstation containing an operation that precedes i and the first workstation containing an operation j such that i precedes j.
3: Move the operation i and all the operations that are linked by an inclusion constraint with operation i from the source workstation to the destination workstation.
4: Run the sequencing algorithm (Algorithm 1) in the source and destination workstations.

Algorithm 4 Principle algorithm for merger move

INPUT: s A feasible solution of the RTLB problem.
OUTPUT: s' A random neighbour of s with respect to merger move

1: Select randomly and uniformly a source workstation.
2: Select randomly and uniformly a destination workstation. This workstation is chosen between the last workstation containing an operation that precedes an operation of the source workstation and the first workstation containing an operation j that is preceded by an operation from the source workstation.
3: Move all the operations of the source workstation to the destination workstation.
4: Delete the source workstation.
5: Run the sequencing algorithm (Algorithm 1) in the destination workstation.

Algorithm 5 Principle algorithm for split move

INPUT: s A feasible solution of the RTLB problem.
OUTPUT: s' A random neighbour of s with respect to insertion move

1: Select randomly and uniformly an operation i (The workstation of i is called S).
2: Partition the workstation S into 2 workstations S_1, S_2: Beginning with the first operation of S, if an operation j is placed before i it is placed with all its inclusive operations in S_1, otherwise it is placed in S_2.
3: Run the sequencing algorithm (Algorithm 1) in the two resulting workstations (S_1 and S_2).

6.3 Simulated annealing

We use an adaptive simulated annealing algorithm to improve the solution returned by I-BFSL. The general scheme of the improvement method is depicted in Algorithm 6. In Algorithm 6, $Cost(X)$ stands for the number of machines used by the solution X.

The adaptive aspect is derived from the following: a random neighbourhood system is chosen among $\{V_1, V_2, V_3\}$ according to the following probability distribution $\{p_1 = \frac{w_1}{w_1+w_2+w_3}, p_2 = \frac{w_2}{w_1+w_2+w_3}, p_3 = \frac{w_3}{w_1+w_2+w_3}\}$ where w_1, w_2

Algorithm 6 Adaptive simulated annealing algorithm

INPUT: X A feasible solution of the RTLB problem given by I-BFSL.
OUTPUT: $Best$ A local optimal solution of the RTLB problem.

$Best := X$
Initialise the temperature T and the weights w_1, w_2 and w_3
while Stop condition of outer loop is not met **do**
  while Stopping condition of inner loop is not met **do**
    repeat
      Select randomly a neighbourhood system $V_i, i \in \{1, 2, 3\}$ with probability $p_i = \frac{w_i}{w_1+w_2+w_3}$
      Generate randomly and uniformly a candidate solution Y: $Y = V_i(X)$
    until Y is feasible
    if $Cost(Y) \leq Cost(X)$ **then**
      Accept: $X = Y$
      if $Cost(Y) < Cost(Best)$ **then**
        $Best := X$
        Increase the weight w_i of V_i
      end if
    else
      Accept with probability $exp(-\frac{Cost(Y)-Cost(X)}{T})$: $X = Y$
    end if
  end while
  Decrease the temperature T
end while

and w_3 are weights associated respectively with V_1, V_2 and V_3. Each time a neighbourhood system V_i gives a strictly improving solution, w_i is relatively increased.

The initialization of the different parameters (temperature and weights), the temperature cooling procedure, the weights update, and the stopping conditions are discussed in the experimentation section.

7 Experimental results

We describe in this section the experimentation being held on a 16Go RAM i7-4790 CPU to evaluate our resolution method. The algorithms were implemented with JAVA 8. IBM's CPLEX (v12.7.0) was used to solve the ILP from the construction phase.

We compare our resolution method with a resolution method suggested from literature: a genetic algorithm that uses either an heuristic or a MIP chromosome decoder [2]: it could also be seen as a matheuristic. The authors did mention on the paper the given objective value for each instance and provided us with the instances sets. The comparison is consequently done by running MATH-BFSL on the same sets of instances:

A1-A15 are instances with 200 operations, they are described in [2]. Those are large-scale problem instances with:

- Number of operations: $n = 200$
- Maximum number of workstations: $s_{max} = 25$

- Maximum number of operations per workstation: $N_{max} = 10$
- Maximum number of machines per workstation: $M_{max} = 5$
- Cycle time: $C = 50$
- Processing times: $d_i \in [1, 10]$
- Setup times: $t_{i,j} \in [0, 2]$
- Number of precedence constraints: $50 \leq |P| \leq 70$
- Number of inclusion and exclusion sets: $7 \leq |I|, |E| \leq 15$
- Number of possible part-fixing positions: $|Pos| = 7$.

The different parameters are determined experimentally:

- Initial temperature is set to 10 and the decreasing scheme is as follows:

$$\begin{cases} T = T - 0.0036 \text{ If } T > 1 \\ T = T - 0.0004 \text{ Otherwise.} \end{cases}$$

- Initial weights w_1, w_1, w_3 are set to 1 and they are increased by adding one if the associated neighbourhood system yields a neighbour strictly improving the best recorded solution.
- We perform δ iterations for each temperature. Experiments are done with different values of δ, $\delta = 1, 20, 50, 100$.

7.1 Construction phase

Table 7 shows the results of I-BFSL (construction phase) for A1-A15 instances. We use the following notations in the table:

- ILP((18)-(32)): denotes the number of machines of the solution returned by the ILP from I-BFSL in the last iteration. The best solution out of 1200" is taken from the linear programming solver (CPLEX).
- I-BFSL: denotes the number of machines of the solution returned by I-BFSL.

Table 4: Initial solutions given by I-BFSL

Instance	ILP((18)-(32))	I-BFSL
A1	24	42
A2	27	30
A3	31	37
A4	24	35
A5	23	42
A6	22	44
A7	23	46
A8	37	38
A9	34	39
A10	26	42
A11	32	37
A12	38	41
A13	23	45
A14	35	36
A15	36	38

We notice from Table 7 that I-BFSL is able to find a feasible solution for all the instances within 1300". The solution given after balancing generation (S1) is sometimes much smaller than the solution returned after balancing generation then operation sequencing (S2).

7.2 Improvement phase

Going from the initial solution given by I-BFSL, the improvement phase is run with different values of δ (number of iterations for each temperature): $\delta = 1$ (Table 8), $\delta = 20$ (Table 9), $\delta = 50$ (Table 10) and $\delta = 100$ (Table 11).

We use the following notations in the tables:

- min: minimum number of machines obtained by the matheuristic over 10 independent runs.
- max: maximum number of machines obtained by the matheuristic over 10 independent runs.
- mean: average number of machines obtained by the matheuristic over 10 independent runs.
- σ: Standard deviation of the number of machines obtained by matheuristic over 10 independent runs.
- (p_1, p_2, p_3): the probability distribution of the neighbourhood systems after the run of the metaheuristic.
- GA[2]: Best solution (with the minimum number of machines) obtained by the genetic algorithm [2] over 10 independent runs.

We notice from Tables 8, 9, 10 and 11 that increasing δ has an impact on results. Indeed, we notice that when δ increases, the min, max, mean and standard deviation values get smaller. However, the improvement of results is at the expense of CPU time. Indeed, experiments with $\delta = 1$ (one iteration for each temperature) only take 25" of CPU time for each instance while experiments with $\delta = 100$ (100 iteration for each temperature) takes about 3000" for each instance.

Small values of the standard deviation (usually smaller than 1) illustrate the robustness of the proposed method.

Table 12 shows the average probability distributions for the neighbourhood systems after the run of MATH-BFSL. We notice that the initial probability distribution is $(p_1, p_2, p_3) = (\frac{1}{3}, \frac{1}{3}, \frac{1}{3})$ then it changes according to the success of each neighbourhood move.

Table 5: Experimentation with $\delta = 1$

	MATH-BFSL			
Instance	min	max	mean	σ
A1	29.0	33.0	30.7	1.1
A2	28.0	30.0	29.4	0.663
A3	27.0	32.0	29.8	1.33
A4	28.0	30.0	29.2	0.6
A5	29.0	31.0	30.3	0.781
A6	28.0	31.0	29.6	0.917
A7	30.0	32.0	31.6	0.663
A8	30.0	32.0	31.0	0.775
A9	28.0	31.0	29.8	0.872
A10	31.0	35.0	33.0	1.1
A11	29.0	31.0	29.8	0.748
A12	30.0	33.0	31.4	0.917
A13	31.0	32.0	31.4	0.49
A14	29.0	32.0	30.8	0.872
A15	29.0	32.0	30.1	0.831

Table 6: Experimentation with $\delta = 20$

	MATH-BFSL			
Instance	min	max	mean	σ
A1	26.0	29.0	27.9	1.04
A2	26.0	27.0	26.4	0.49
A3	26.0	28.0	26.6	0.663
A4	25.0	27.0	26.1	0.539
A5	26.0	28.0	27.2	0.748
A6	26.0	28.0	26.8	0.6
A7	28.0	29.0	28.3	0.458
A8	27.0	28.0	27.8	0.4
A9	26.0	28.0	26.8	0.6
A10	29.0	31.0	29.8	0.748
A11	26.0	28.0	26.9	0.539
A12	27.0	28.0	27.8	0.4
A13	28.0	29.0	28.7	0.458
A14	27.0	28.0	27.2	0.4
A15	26.0	28.0	27.2	0.6

Table 9: Average probability distributions of neighbourhood systems after the run of MATH-BFSL

	p_1, p_2, p_3			
Instance	$\delta = 1$	$\delta = 20$	$\delta = 50$	$\delta = 100$
A1	(0.75, 0.16,0.088)	(0.76,0.17,0.076)	(0.78,0.15,0.075)	(0.78,0.14,0.073)
A2	(0.35,0.33,0.33)	(0.38,0.33,0.29)	(0.39,0.33,0.28)	(0.41,0.33,0.26)
A3	(0.49,0.38,0.13)	(0.6,0.29,0.1)	(0.61,0.29,0.1)	(0.61,0.29,0.096)
A4	(0.58,0.26,0.15)	(0.64,0.24,0.12)	(0.65,0.23,0.12)	(0.67,0.21,0.11)
A5	(0.76,0.16,0.081)	(0.79,0.14,0.072)	(0.78,0.15,0.069)	(0.81,0.12,0.067)
A6	(0.77,0.16,0.069)	(0.8,0.14,0.061)	(0.8,0.14,0.059)	(0.8,0.14,0.058)
A7	(0.8,0.14,0.066)	(0.82,0.12,0.058)	(0.83,0.11,0.056)	(0.83,0.11,0.057)
A8	(0.64,0.23,0.13)	(0.66,0.23,0.1)	(0.69,0.2,0.1)	(0.67,0.23,0.1)
A9	(0.57,0.33,0.1)	(0.6,0.32,0.085)	(0.6,0.31,0.083)	(0.61,0.31,0.081)
A10	(0.78,0.11,0.1)	(0.8,0.11,0.089)	(0.79,0.12,0.086)	(0.82,0.094,0.084)
A11	(0.56,0.31,0.13)	(0.65,0.25,0.1)	(0.65,0.25,0.098)	(0.67,0.23,0.095)
A12	(0.54,0.36,0.1)	(0.6,0.32,0.082)	(0.61,0.31,0.082)	(0.63,0.29,0.079)
A13	(0.77,0.16,0.069)	(0.8,0.14,0.062)	(0.8,0.14,0.059)	(0.81,0.13,0.058)
A14	(0.39,0.46,0.16)	(0.47,0.41,0.13)	(0.45,0.43,0.12)	(0.49,0.39,0.11)
A15	(0.51,0.37,0.12)	(0.59,0.31,0.099)	(0.62,0.29,0.095)	(0.6,0.31,0.094)

Table 7: Experimentation with $\delta = 50$

	MATH-BFSL			
Instance	min	max	mean	σ
A1	27.0	29.0	27.6	0.8
A2	25.0	27.0	25.9	0.539
A3	25.0	26.0	25.8	0.4
A4	25.0	27.0	25.9	0.539
A5	26.0	27.0	26.4	0.49
A6	25.0	27.0	26.1	0.539
A7	26.0	28.0	27.3	0.64
A8	27.0	28.0	27.3	0.458
A9	26.0	27.0	26.3	0.458
A10	28.0	30.0	29.1	0.539
A11	25.0	27.0	26.3	0.64
A12	26.0	28.0	27.3	0.64
A13	27.0	29.0	28.1	0.539
A14	26.0	28.0	26.5	0.671
A15	26.0	27.0	26.4	0.49

Table 8: Experimentation with $\delta = 100$

	MATH-BFSL			
Instance	min	max	mean	σ
A1	26.0	28.0	26.9	0.539
A2	25.0	26.0	25.6	0.49
A3	25.0	27.0	25.5	0.671
A4	25.0	26.0	25.7	0.458
A5	25.0	27.0	26.3	0.64
A6	25.0	26.0	25.6	0.49
A7	27.0	28.0	27.4	0.49
A8	27.0	28.0	27.2	0.4
A9	26.0	27.0	26.2	0.4
A10	28.0	30.0	28.8	0.6
A11	25.0	26.0	25.4	0.49
A12	27.0	28.0	27.2	0.4
A13	27.0	28.0	27.5	0.5
A14	26.0	27.0	26.4	0.49
A15	25.0	26.0	25.9	0.3

Initially, the neighbourhood systems have the same probability to be chosen in the balancing perturbation. Table 12 shows that the neighbourhood system V_1 has the greatest probability to be chosen then V_2. The probability of V_3 is close to 0. Since in our algorithm the probabilities are only increased if the neighbourhood system yields a neighbour strictly improving the best recorded solution, the table shows that neighbourhood system V_1 then V_2 are the most efficient for improving a solution. V_3 almost never improves a solution, its purpose is above all to introduce diversity into the search process.

Table 10: Comparison between the matheuristic, the genetic algorithm of the literature and the linear relaxation lower bound

	GA (min)	I-BFSL	MATH-BFSL (min)			
Instance			$\delta = 1$	$\delta = 20$	$\delta = 50$	$\delta = 100$
A1	33	42	29	**26**	27	**26**
A2	33	30	28	26	**25**	**25**
A3	31	37	27	26	**25**	**25**
A4	29	35	28	**25**	**25**	**25**
A5	32	42	29	26	26	**25**
A6	32	44	28	26	**25**	**25**
A7	34	46	30	28	**26**	27
A8	31	38	30	**27**	**27**	**27**
A9	30	39	28	**26**	**26**	**26**
A10	32	42	31	29	**28**	**28**
A11	30	37	29	26	**25**	**25**
A12	31	41	30	27	**26**	27
A13	33	45	31	28	**27**	**27**
A14	31	36	29	27	**26**	**26**
A15	33	38	29	26	26	**25**

Table 13 compares the performance of the suggested matheuristic with the genetic algorithm from literature [2]. In this last, only the minimum cost over 10 independent runs is reported. Therefore, we report the min over 10 independent runs for MATH-BFSL for different values of δ.

Table 13 gives a clear advantage to MATH-BFSL compared to the genetic algorithm of the literature [2]. Indeed, the construction phase (I-BFSL) usually gives a solution of cost superior to the one of the genetic algorithm but this solution is rapidly improved with the improvement phase (MATH-BFSL). The results of the genetic algorithm are already outperformed with $\delta = 1$ which runs in 25". With $\delta = 100$ (3000"), the improvement is even more drastic compared to literature and approaches 20% for some instances. Besides, even the max of MATH-BFSL is lower than the min of GA for some instances. However, we must say that this very advantageous results are obtained at the expense of CPU time. Indeed, the genetic algorithm runs in 900" whereas the constructive phase (I-BFSL) runs in 1200" and the improvement phase CPU time varies from 25" (for $\delta = 1$) to 3000" (for $\delta = 100$) as shown in Table 14. MATH-BFSL has a greater CPU time than the genetic algorithm from literature because it embeds several components from mathematical programming (integer linear programming, constraint generation and dynamic programming). These components are an important source of the good performance of the proposed method. Table 14 reports the average CPU times of the algorithms.

Table 11: Averge approximate CPU times on A1-A15 instances

Algorithm	Average approximate CPU time
I-BFSL	1300”
Improvement phase of MATH-BFSL with $\delta = 1$	25”
Improvement phase of MATH-BFSL with $\delta = 20$	1000”
Improvement phase of MATH-BFSL with $\delta = 50$	1700”
Improvement phase of MATH-BFSL with $\delta = 100$	3000”
Genetic algorithm from literature [2]	900”

8 Conclusion and perspectives

Matheuristics are efficient optimization procedures that take advantage simultaneously of the accuracy of exact methods and the scalability of heuristics. We suggested in this article a matheuristic with feasibility and performance guaranty. The proposed method lies on a constructive phase and an improvement phase. It contains several components from exact methods (linear programming, constraint generation and dynamic programming) and metaheuristics (simulated annealing). The experiments show that the approach outperforms a genetic algorithm from literature.

Several directions could be taken following this research:

- Investigating the symmetric approach (Sequence First, Balance Last) that consists of balancing the line given an overall sequence of all operations could constitute a promising track of research.
- Studying multiple products in the multi-model context raises a rebalancing problem and could be an interesting direction for future research.

Acknowledgements The authors acknowledge the support received from the *Agence Nationale de la Recherche* of the French government through the program ”Investissements d Avenir”(16-IDEX-0001 CAP 20-25).

References

1. Battaïa, O., Dolgui, A.: A taxonomy of line balancing problems and their solution approaches. International Journal of Production Economics **142**(2), 259 – 277 (2013)
2. Borisovsky, P.A., Delorme, X., Dolgui, A.: Genetic algorithm for balancing reconfigurable machining lines. Computers & Industrial Engineering **66**(3), 541 – 547 (2013). Special Issue: The International Conferences on Computers and Industrial Engineering (ICC&IEs) - series 41
3. Essafi, M., Delorme, X., Dolgui, A.: Balancing lines with CNC machines: A multi-start ant based heuristic. CIRP Journal of Manufacturing Science and Technology **2**(3), 176–182 (2010)
4. Essafi, M., Delorme, X., Dolgui, A.: Balancing machining lines: a two-phase heuristic. Studies in Informatics and Control **19**(3), 243–252 (2010)
5. Essafi, M., Delorme, X., Dolgui, A.: A reactive grasp and path relinking for balancing reconfigurable transfer lines. International Journal of Production Research **50**(18), 5213–5238 (2012)

6. Essafi, M., Delorme, X., Dolgui, A., Guschinskaya, O.: A MIP approach for balancing transfer line with complex industrial constraints. Computers & Industrial Engineering **58**(3), 393 – 400 (2010)
7. Held, M., Karp, R.M.: A dynamic programming approach to sequencing problems. Journal of the Society for Industrial and Applied Mathematics **10**(1), 196–210 (1962)
8. Koren, Y., Heisel, U., Jovane, F., Moriwaki, T., Pritschow, G., Ulsoy, G., Van Brussel, H.: Reconfigurable manufacturing systems. CIRP annals **48**(2), 527–540 (1999)
9. Koren, Y., Shpitalni, M.: Design of reconfigurable manufacturing systems. Journal of Manufacturing Systems **29**(4), 130 – 141 (2010)
10. Koren, Y., Wang, W., Gu, X.: Value creation through design for scalability of reconfigurable manufacturing systems. International Journal of Production Research **55**(5), 1227–1242 (2017)
11. Lahrichi, Y., Deroussi, L., Grangeon, N., Norre, S.: Reconfigurable transfer line balancing problem: A new MIP approach and approximation hybrid algorithm. In: MOSIM 2018 (Modélisation et Simulation). Toulouse, France (2018)
12. Salii, Y.: Revisiting dynamic programming for precedence-constrained traveling salesman problem and its time-dependent generalization. European Journal of Operational Research **272**(1), 32–42 (2019)
13. Salveson, M.E.: The assembly line balancing problem. The Journal of Industrial Engineering pp. 18–25 (1955)
14. Van Laarhoven, P.J., Aarts, E.H.: Simulated annealing. In: Simulated annealing: Theory and applications, pp. 7–15. Springer (1987)