A new split-based hybrid metaheuristic for the Reconfigurable Transfer Line Balancing Problem

Y Lahrichi, Laurent Deroussi, Nathalie Grangeon, Sylvie Norre

A new split-based hybrid metaheuristic for the Reconfigurable Transfer Line Balancing Problem

Y. Lahrichi, L. Deroussi, N. Grangeon, S. Norre

LIMOS CNRS UMR 6158 - 1 Rue de la Chebarde, 63178 Aubière, France

ARTICLE HISTORY
Compiled December 11, 2019

ABSTRACT
We consider the Reconfigurable Transfer Line Balancing Problem. This 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. The machines considered are mono-spindle head CNC machines which may imply **sequence-dependent** setup times between operations in order to perform tool changes. Therefore, the operations allocated to a workstation should be sequenced. Besides, accessibility, inclusion, exclusion and precedence constraints between operations are considered. In this article, we propose a polynomial exact algorithm that balances the transfer line provided the overall sequence of the operations (called "giant sequence") is given. We use this algorithm to solve the balancing problem when the overall sequence of operations is not fixed by embedding it in a metaheuristic framework. We perform experimentation on literature instances. The results obtained show the effectiveness of the proposed approach compared to literature.

KEYWORDS
Split algorithm; metaheuristic; balancing; transfer line; **setup time**; polynomial algorithm; **sequencing**.

1. Introduction

New consuming trends, global competition and growing variety in demand in the actual economical context raise an important issue in transfer line design. Shortening life cycle times imposes the consideration of reconfigurability ([Koren and Shpitalni(2010)], [Mehrabi et al.(2002)]). The modern transfer lines should be easily and cost-effectively reconfigurable to address two different issues: the variability in production size and the variability in the product specifications.

A Reconfigurable Transfer Line (RTL) could be seen as a serial line of workstations. Each workstation is equipped by multiple machines operating in parallel. The RTL is paced and every part is delivered to a single machine in every workstation. The machines from the same workstation perform the same sequence of operations.

The RTL highly addresses the issue of production size variability. Indeed the ability to add or remove a machine in a workstation allows monitoring the cycle time with high granularity which is **referred** to as scalability [Koren, Wang, and Gu(2017)].

The RTL we consider is equipped with mono-spindle head CNC machines. Those machines can perform a large set of operations, each machine being equipped with

a tool magazine. To perform an operation, a machine needs a specific tool. Thus, **sequence-dependent** setup times between operations must be considered in addition to processing times in order to perform tool changing.

Our objective is to study the balancing of such lines in the presence of several types of products and in particular the changes of production campaigns between two types of products. Before considering this global problem, we focused on the problem of balancing such lines in the presence of a single product. This is the problem we are dealing with in this article.

We define the problem and introduce the notations used in the remainder of the paper in section 2. The related work is described in section 3. We **propose** a polynomial algorithm to balance a sequence of operations (subproblem) in section 4 and incorporate it in a metaheuristic to solve the general problem in section 5. An experimental study is covered in section 6 of the paper.

2. The Reconfigurable Transfer Line Balancing Problem

2.1. *Description of the problem and notations*

The Reconfigurable Transfer Line Balancing problem (RTLB) is a combinatorial optimisation problem whose instance is described by the following data:

- A set of operations, the corresponding processing times and setup times.
- A maximum number of workstations to be used.
- A maximum number of machines per workstation.
- A cycle time.
- A maximum number of operations to be allocated to a workstation.
- Precedence, inclusion, exclusion and accessibility data.

The optimisation problem consists then in allocating the operations to the workstations, sequencing the operations in each workstation and determining a number of machines per workstation while minimising the overall number of machines used and respecting the following constraints:

- 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.
- 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 must be processed before j.
- 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.
- 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 opera-**

tions (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 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.

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

Table 1. The notations.

n	Number of operations
N	Set of operations, indexed on $\{1, 2, \ldots, n\}$
s_{max}	Maximum number of workstations
S	Set of workstations, indexed on $\{1, 2, \ldots, s_{max}\}$
P	Set of couples $(i, j) \in N^2$ such that i precedes j
M_{max}	Maximum number of machines in a workstation
N_{max}	Maximum number of operations assigned to 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 a workstation
I	Set of couples $(i, j) \in N^2$ linked with an inclusion constraint
E	Set of **subsets 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. *Mathematical formulation*

In order to clarify the definition of the problem, we describe the ILP proposed in [Lahrichi et al.(2018)]. The approach is based on modelling the sequences **of operations assigned to each** workstation. It uses the following binary variables:

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

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

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

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

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

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

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

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

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

$$\sum_{s=1}^{s_{max}} \sum_{j=1}^{N_{max}} x_{i,s,j} = 1, \forall i \in N \quad (1)$$

$$\sum_{i=1}^{n} x_{i,s,j} \leq 1, \forall s \in S, \forall j \in \{1, \ldots, N_{max}\} \quad (2)$$

$$\sum_{i=1}^{n} x_{i,s,j+1} \leq \sum_{i=1}^{n} x_{i,s,j}, \forall s \in S, \forall j \in \{1, \ldots, N_{max} - 1\} \quad (3)$$

$$\sum_{k=1}^{M_{max}} v_{s,k} = y_s, \forall s \in S \quad (4)$$

$$y_{s+1} \leq y_s, \forall s \in \{1, \ldots, s_{max} - 1\} \quad (5)$$

$$\sum_{s=1}^{s_{max}} \sum_{j=1}^{N_{max}} (N_{max}.(s-1) + j) x_{i,s,j} \leq \sum_{s=1}^{s_{max}} \sum_{j=1}^{N_{max}} (N_{max}.(s-1) + j) x_{i',s,j}, \forall (i, i') \in P \quad (6)$$

$$\sum_{i=1}^{n}\sum_{j=1}^{N_{max}} d_i.x_{i,s,j} + \sum_{i=1}^{n}\sum_{i'=1}^{n} t_{i,i'}.z_{i,i',s} \leq C.\sum_{k=1}^{M_{max}} k.v_{s,k}, \forall s \in S \quad (7)$$

$$x_{i,s,j} + x_{i',s,j+1} \leq 1 + z_{i,i',s}, \forall (i,i') \in N^2, i \neq i', \forall j \in \{1,\ldots,N_{max}-1\}, \forall s \in S \quad (8)$$

$$x_{i,s,j} - \sum_{i' \in N; i' \neq i} x_{i',s,j+1} \leq w_{i,s}, \forall i \in N, \forall s \in S, \forall j \in \{1,\ldots,N_{max}-1\} \quad (9)$$

$$x_{i,s,N_{max}} \leq w_{i,s}, \forall i \in N, \forall s \in S \quad (10)$$

$$w_{i,s} + x_{i',s,1} \leq 1 + z_{i,i',s}, \forall (i,i') \in N^2, i \neq i', \forall s \in S \quad (11)$$

$$\sum_{s=1}^{s_{max}}\sum_{j=1}^{N_{max}} s.x_{i,s,j} = \sum_{s=1}^{s_{max}}\sum_{j=1}^{N_{max}} s.x_{i',s,j}, \forall (i,i') \in I \quad (12)$$

$$\sum_{i \in ES}\sum_{j=1}^{N_{max}} x_{i,s,j} \leq |ES| - 1, \forall ES \in E, \forall s \in S \quad (13)$$

$$\sum_{a \in Pos} u_{s,a} \leq 1, \forall s \in S \quad (14)$$

$$\sum_{j=1}^{N_{max}} x_{i,s,j} - \sum_{a \in Pos_i} u_{s,a} \leq 0, \forall i \in N, \forall s \in S \quad (15)$$

2.3. *Example*

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

- The part requires the execution of 7 operations numbered from 1 to 7 ($n = 7$).
- At most **5** workstations can be used ($s_{max} = 5$).
- Precedence constraints are given by:

$$P = \{(1,3),(2,3),(3,4),(4,5),(5,6),(5,7)\}$$

  represented by figure 1.

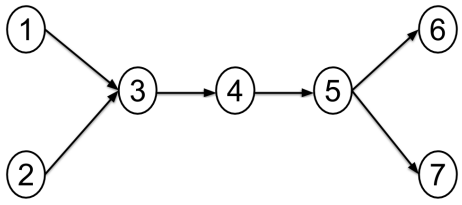

Figure 1. Precedence graph.

- $N_{max} = 3$, Maximum number of operations that could be assigned to a workstation.
- $M_{max} = 3$, Maximum number of machines that could be hosted by a workstation.
- Processing times are represented in Table 2.
- Setup times are represented in Table 3.

Table 2. Processing times.

i	1	2	3	4	5	6	7
d_i	1.5	1	3.5	1.5	2.5	3	1

Table 3. Setup times.

$t_{i,j}$	$j=1$	2	3	4	5	6	7
$i=1$	0	0.5	1	1	1	1	1
2	1	0	0.5	1	1	1	1
3	0.5	1	0	1	1	1	1
4	1	1	1	0	0.5	1	1
5	1	1	1	0.5	0	1	1
6	1	1	1	1	1	0	0.5
7	1	1	1	1	1	0.5	0

- $\mathbf{C} = 2.5$, cycle time.
- Inclusion and exclusion constraints are given by:

$$I = \{(1,2)\}, E = \{\{5,6\}\}$$

- Accessibility constraints are given as follows: $Pos = \{1,2,3,4\}$, $Pos_4 = \{1,2\}$, $Pos_5 = \{3,4\}$, $Pos_i = \{1,2,3,4\}, \forall i \in \{1,2,3,6,7\}$

A feasible solution is represented in figure 2. $\lceil \frac{d_1+d_2+t_{1,2}+t_{2,1}}{C} \rceil = \lceil \frac{4}{2.5} \rceil = 2$, so 2 machines are required for workstation 1. The machines that must be hosted by the other workstations are indicated in figure 2.

3. State of the art

The problem studied in this paper could be seen as a line balancing problem. This problem has been extensively studied in the literature. A taxonomy of this problem could be found in [Battaïa and Dolgui(2013)].

The particularity of the RTLB problem is to consider simultaneously parallel machines, setup times and transfer line environment constraints (inclusion, exclusion and accessibility). These three components are most often studied separately in literature:

- Parallel machines in the workstations in order to reduce the cycle time: The problem is known as the simple assembly line balancing problem with parallel workstations. It has been introduced in [Buxey(1974)] and dealt with in [Vilarinho and Simaria(2006)] and [Rabbani et al.(2016)] in the case of mixed-model production lines.
- Setup times on machines between operations: The problem is known as the sequence-dependent assembly line balancing problem (SDALBP). It has been defined in [Andrés, Miralles, and Pastor(2008)]. The problem is studied in [Martino and Pastor(2010)].
- Inclusion, exclusion and accessibility constraints: The constraints come from the machining industry. Balancing problems considering those constraints are known as transfer line balancing problems. **Firstly introduced in [Dolgui, Guschinsky, and Levin(2000)], some other authors consider those constraints** [Battaïa et al.(2012)]. **More references could be found in** [Battaïa and Dolgui(2013)].

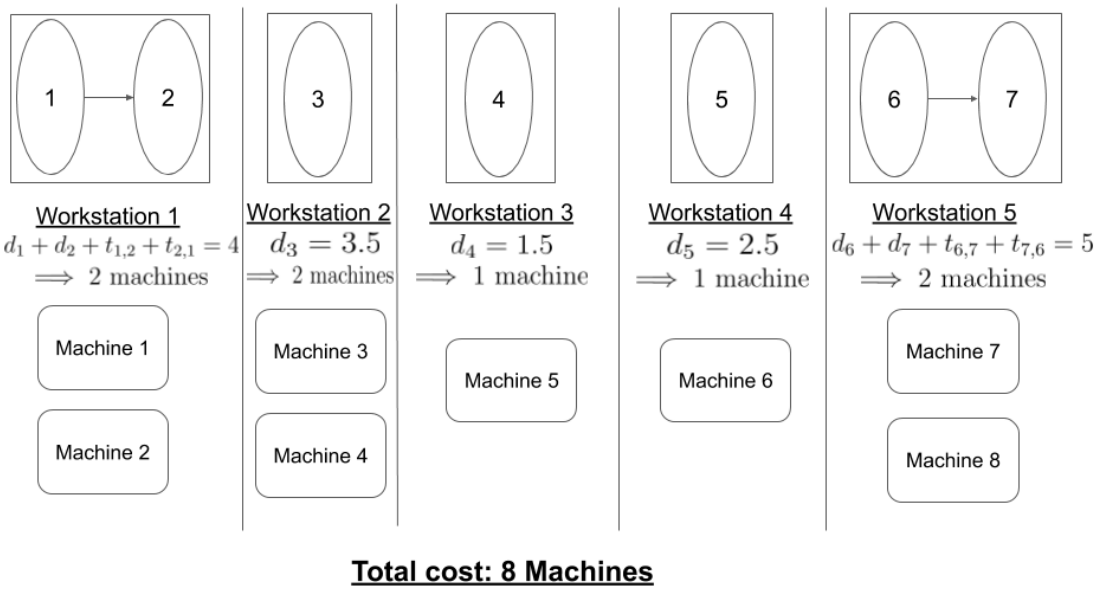

Figure 2. Example of feasible solution.

The RTLB problem consists of simultaneously solving two **sub**problems:

- A balancing **sub**problem: assign the operations to the workstations.
- A sequencing **sub**problem: sequence the operations assigned to each workstation.

Therefore, resolution approaches for the RTLB problem could be classified within three categories:

- **Integrated approaches:** the balancing and sequencing steps are performed simultaneously. Most literature methods are of this type. [Essafi et al.(2010)] suggest a MIP approach while [Borisovsky, Delorme, and Dolgui(2012)] use a set partitioning model. We also cite [Essafi, Delorme, and Dolgui(2010b)] (a two phase heuristic) and [Essafi, Delorme, and Dolgui(2010a)] (Ant colony optimisation metaheuristic) as integrated approaches.
- **Balance-First Sequence-Last (BFSL) approaches:** the sequencing step is done after the balancing step. We proposed an approximation algorithm of this type in [Lahrichi et al.(2018)]. To the best of our knowledge no other approach of type BFSL has been suggested.
- **Sequence-First Balance-Last (SFBL) approaches:** the balancing step is done after the sequencing step. The problem of balancing a sequence of operations in a reconfigurable transfer line refers to solving the RTLB problem given an overall sequence of all operations (giant sequence). This balancing subproblem consists in determining optimal positions (of the giant sequence) where to split the giant sequence. Each subsequence thus obtained is allocated to a different workstation in the order. The subproblem was dealt with in the literature by means of ILP ([Borisovsky, Delorme, and Dolgui(2012)], [Delorme, Malyutin, and Dolgui(2016)]) and heuristics ([Borisovsky, Delorme, and Dolgui(2012)]).

  In [Borisovsky, Delorme, and Dolgui(2012)], a solution is represented as a giant sequence of all operations, either a heuristic decoder or a MIP is suggested

to evaluate it. In [Delorme, Malyutin, and Dolgui(2016)], a multi-objective algorithm is proposed to simultaneously minimise cycle time and line cost (number of workstations × cost of a workstation + number of machines × cost of a machine). The algorithm uses a giant sequence of operations then splits it using the MIP from [Borisovsky, Delorme, and Dolgui(2012)] minimising the line cost.

The literature study allowed us to note, that to the best of our knowledge, there is no exact polynomial algorithm that allows to compute a "balancing" for a given giant sequence. **Indeed, this subproblem was only solved by means of an heuristic and a MIP ([Borisovsky, Delorme, and Dolgui(2012)], [Delorme, Malyutin, and Dolgui(2016)]).** Our contribution is therefore at this level: a polynomial exact algorithm (called "split") to compute the optimal balancing given a giant sequence. We further **embed** this exact polynomial algorithm in a metaheuristic (**ILS**) to solve the general problem (RTLB).

4. A polynomial algorithm to solve the balancing subproblem

Given a *giant sequence* of operations, we consider the subproblem of balancing the line respecting the giant sequence: **it consists in splitting the giant sequence into different subsequences. Each subsequence thus obtained is allocated to a workstation so that the i^{th} subsequence is allocated to the i^{th} workstation. The split is done while minimising the total number of machines required to respect all the constraints.**

Given a single giant sequence, more than one balancing solution could exist. For example, we suppose the giant sequence is: $1, 2, 3, 4, 5, 6, 7$ for the example given in (subsection 2.3). Two balancing solutions respecting the giant sequence (as well as all the problem constraints) could be:

- Assigning the subsequence $1, 2$ to the first workstation, 3 to the second workstation, 4 to the third workstation, 5 to the fourth workstation and $6, 7$ to the fifth workstation: **this solution could be denoted [1,2|3|4|5|6,7] (A pipe (|) indicates a workstation change).** This solution **requires** 8 machines **and is depicted in Figure 2**.
- Assigning the subsequence $1, 2, 3$ to the first workstation, 4 to the second workstation, 5 to the third workstation and $6, 7$ to the fourth workstation. **We can denote it : [1,2,3|4|5|6,7].** This solution **requires** 7 machines (we will **see** later that it is optimal **for this sequence** as represented in Figure 4).

This section is devoted to the description of the split algorithm, in order to optimally solve the balancing subproblem (**minimising the number of machines used**). We first model the balancing subproblem as a constrained shortest path and construct the underlying graph (subsection 4.1). We introduce afterwards a polynomial algorithm (split) to solve **this** subproblem (subsection 4.2).

4.1. *Construction of the graph*

Given an instance $\mathcal{I}$ of the RTLB, we suppose **for sake of simplicity and** without loss of generality that the giant sequence is "1,2,...,n".

Solving the balancing subproblem minimising the number of machines is equivalent to find the shortest weighted path of length lower than or equal to s_{max} arcs between

fictitious vertex 0 and the vertex n in the directed weighted graph:

$$G = (V, A), V = \{0, 1, 2, .., n\}, A = \{(i, j), i < j\}$$

Vertex 0 is a fictitious vertex. Other vertices ({1,2,..,n}) model the operations. The arc (i, j) models the fact that the operations $\{i + 1, ..., j\}$ are assigned to the same workstation in every path from 0 to n taking path through (i, j).

Weights on arcs are given by:

$$c_{i,j} = \left\lceil \frac{\sum_{k=i+1}^{j} d_k + (\sum_{k=i+1}^{j-1} t_{k,k+1}) + t_{j,i+1}}{C} \right\rceil$$

which could be interpreted as the number of machines necessary to perform the sequence $i + 1,\ldots,j$.

By way of illustration, using the data of the example described in section 2.3., we obtain:

$$c_{0,3} = \left\lceil \frac{d_1 + d_2 + d_3 + t_{1,2} + t_{2,3} + t_{3,1}}{C} \right\rceil = \left\lceil \frac{7.5}{2.5} \right\rceil = 3$$

i.e 3 machines are needed to assign the subsequence "1,2,3" to a workstation.

Some of the arcs must be deleted because some constraints are violated. All the cases are listed below:

- **The maximum number of operations per workstation is exceeded:** An arc (i, j) violating the maximum number of operations per workstation constraint can be detected by the following condition:

  $$j - i > N_{max}$$

  For example: $(0, 4) \notin A$ because $4 - 0 = 4 > N_{max} = 3$.
- **The maximum number of machines per workstation is exceeded:** Likewise, an arc (i, j) violating the maximum number of machines per workstation constraint can be detected by the following condition:

  $$c_{i,j} = \left\lceil \frac{\sum_{k=i+1}^{j} d_k + (\sum_{k=i+1}^{j-1} t_{k,k+1}) + t_{j,i+1}}{C} \right\rceil > M_{max}$$

  For example: $(2, 5) \notin A$ because $c_{2,5} = 4 > M_{max}$.
- **Inclusion constraints are not satisfied:** An arc (i, j) violating the inclusion constraints can be detected by the following condition:

  $$\exists (a, b) \in I \ / \ \mathbb{1}_{\{i+1,\ldots,j\}}(a)^1 + \mathbb{1}_{\{i+1,\ldots,j\}}(b) = 1$$

  For example: $(0, 1) \notin A$ because $\{1, 2\} \in I$.

[1] $\mathbb{1}_X(a)$ **is the indicator function, equals 1 if** $a \in X$ **and 0 otherwise.**

- **Exclusion constraints are not satisfied:** An arc (i, j) violating the exclusion constraints can be detected by the following condition:

$$\exists ES \in E \ / \ \sum_{a \in ES} \mathbb{1}_{\{i+1,\ldots,j\}}(a) = |ES|$$

  For example: $(4, 6) \notin A$ because $\{5, 6\} \subset E$.
- **Accessibility constraints are not satisfied:** An arc (i, j) violating the accessibility constraints can be detected by the following condition:

$$\cap_{k \in \{i+1,\ldots,j\}} Pos_k = \emptyset$$

  For example: $(3, 6) \notin A$ because $Pos_4 \cap Pos_5 \cap Pos_6 = \emptyset$.

Example 4.1. An illustration of the graph is given in figure 3 for the instance given in section 2 and giant sequence: "1,2,3,4,5,6,**7**". Arcs violating constraints were deleted. For example, arcs (0,1), (1,2) and (1,3) are deleted due to the violation of the inclusion constraint "(1,2)". The arc (3,5) is deleted due to the violation of the accessibility constraint. Arc (4,6) is deleted due to the violation of the exclusion constraint "{5,6}". Arcs (0,4), (0,5), (0,6) and (0,7) are deleted due to exceeding the maximum number of machines (M_{max}) and operations (N_{max}) per workstation.

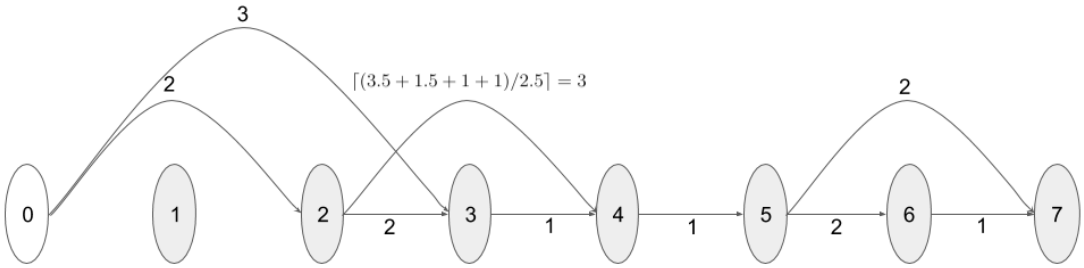

Figure 3. *Final graph for the example instance*

The graph could be disconnected despite the existence of a balancing solution: in the example of figure 3, the disconnectivity comes from the fact that operations 1 and 2 **must** be assigned to the same workstation due to the inclusion constraint.

4.2. *A polynomial algorithm to solve the constrained shortest path problem: split*

Given an instance $\mathcal{I}$ of RTLB problem and a giant sequence $\sigma = (\sigma_1, \sigma_2, \ldots, \sigma_n)$, we denote the above-mentioned graph $\mathcal{H}_{\mathcal{I}}(\sigma)$. **Finding an optimal balancing respecting the giant sequence σ is equivalent to finding the shortest path between the two extremities of the graph $\mathcal{H}_{\mathcal{I}}(\sigma)$, i.e. 0 and σ_n (0 and 7 in Figure 3), of length lower than (or equal to) s_{max} arcs (i.e. the shortest path constrained not to exceed s_{max} arcs). This is because an arc stands for a workstation and the solution cannot use more than s_{max} workstations. The existence of such**

a path between the starting node and the ending node implies the feasibility of the giant sequence. If there is no path **having s_{max} arcs or less** between **0 and σ_n in $\mathcal{H}_{\mathcal{I}}(\sigma)$, then** there exists no feasible balancing solution **respecting the giant sequence σ.**

Moreover, we must note that the more the problem is constrained, the less arcs are contained in the graph and the more the problem is easy to solve.

The shortest path approach to split a giant sequence is well known in solving Vehicle Routing problems (VRP). Indeed, [Beasley(1983)] was the first to prove that the problem of finding a VRP solution respecting some giant tour is polynomial and equivalent to a shortest path problem in some appropriate graph. Afterwards, some researches were conducted where the split is embedded in metaheuristics [Prins, Lacomme, and Prodhon(2014)]. We propose here an original way to adapt ideas from VRP to balancing problems.

We propose the split algorithm (algorithm 1) in order to compute a **constrained shortest path. It cannot be solved with classic shortest path algorithms because those algorithms do not take into account the maximum number of arcs in the path. The split algorithm could be seen as an adaptation of Bellman-Ford algorithm to solve the problem of finding a shortest path constrained not to exceed s_{max} arcs in the graph $\mathcal{H}_{\mathcal{I}}(\sigma)$.** It uses labels on nodes to encompass information on the system state. **The labels keep track of the number of machines and the number of arcs.**

A label l is represented by a couple:

$$l = (a, b)$$

where a denotes the cost (number of machines) used by the path represented by the label l and b denotes the number of workstations used by this path (i.e. the number of arcs in the path). **The algorithm uses a dominance rule in order to avoid a combinatorial explosion.** We define a set of labels L_i for node i. The dominance rule limits the number of labels per node to s_{max}. Every label in L_i **corresponds to** a path (partial solution) between 0 and i.

Definition 4.2. (Dominance rule) (a, b) is dominated by (a', b') if:

$$a' \leq a \text{ and } b' \leq b$$

Algorithm 1 starts with fictitious node 0 labelled $L_0 := \{(0, 0)\}$ and continues with the other nodes following the giant sequence. For every node t and every label $(a_t, b_t) \in L_t$, the algorithm explores every outgoing arc (t, i) and tries to propagate it (i.e add a label to the list of labels of node i denoted L_i) if:

$$(a_t + c_{t,i}, b_t + 1) \text{ is not } \textit{dominated} \text{ by a label of } L_i \text{ (Propagation rule)}$$

If so, the label $(a_t+c_{t,i}, b_t+1)$ is added to L_i and all labels dominated by $(a_t+c_{t,i}, b_t+1)$ are deleted from L_i. The shortest path cost is stored in C^*.

Lemma 4.3. *The dominance rule limits the number of labels per node to s_{max}.*

Proof. Suppose by contradiction that we have more than s_{max} labels for some node i.

Algorithm 1 *split*

INPUT ($\mathcal{I}$,σ) where $\mathcal{I}$ is an instance of the RTLB problem and σ is a giant sequence respecting precedence constraints. We suppose without loss of generality that $\sigma = \{1, 2, ..., n\}$

OUTPUT S: An optimal solution (with the minimal number of machines C^*) respecting σ if there exists a feasible solution

Build the graph $\mathcal{H}_{\mathcal{I}}(\sigma)$
$L_0 := \{(0, 0)\}$
for t=1 **to** n **do**
  $L_t := \emptyset$
end for
for t=0 **to** n-1 **do**
  for all $i/(t, i) \in A$ *(Propagate labels from* L_t*)* **do**
    for all $(a_t, b_t) \in L_t$ **do**
      if ($b_t < s_{max} - 1$ or $i = n$) **then**
        if $(a_t + c_{t,i}, b_t + 1)$ is not *dominated* by an element of L_i **then**
          $L_i := L_i \cup \{(a_t + c_{t,i}, b_t + 1)\}$
          if $(a_t + c_{t,i}, b_t + 1)$ dominates some element $(a_i, b_i) \in L_i$ **then**
            $L_i := L_i \backslash \{(a_i, b_i)\}$
          end if
        end if
      end if
    end for
  end for
end for
if $L_n \neq \emptyset$ **then**
  $C^* := Min_{(a_i,b_i)\in L_n}(a_i)$
  Decode the path of cost C^* to build S
end if

Then, necessarily there must exist two labels $(a, b), (a', b) \in L_i$ (i.e with same number of workstations), because for every label (x, y) we have:

$$y \in \{1, 2, \ldots, s_{max}\}$$

The coexistence of (a, b) and (a', b) is impossible due to the dominance rule.

□

Theorem 4.4. *The algorithm runs in $O(n^4)$ where n is the number of operations.*

Proof. The algorithm performs dominance tests for each 3-uplet (label of origin node, arc, label of destination node). Thus, it runs in $O(m.s_{max}^2)$ where m is the number of arcs in the graph. Since $s_{max} \leq n$ and $m \leq n + (n-1) + ... + 1 = \frac{n(n+1)}{2}$, split runs in $O(n^4)$. □

Example 4.5. In Figure 4, we show how the labels are propagated on our example. The labels of each node are written below it in the order they are **computed** by the algorithm. **A** crossed label represent**s a** dominated label. The optimal solution is highlighted: it uses 7 machines and 4 workstations.

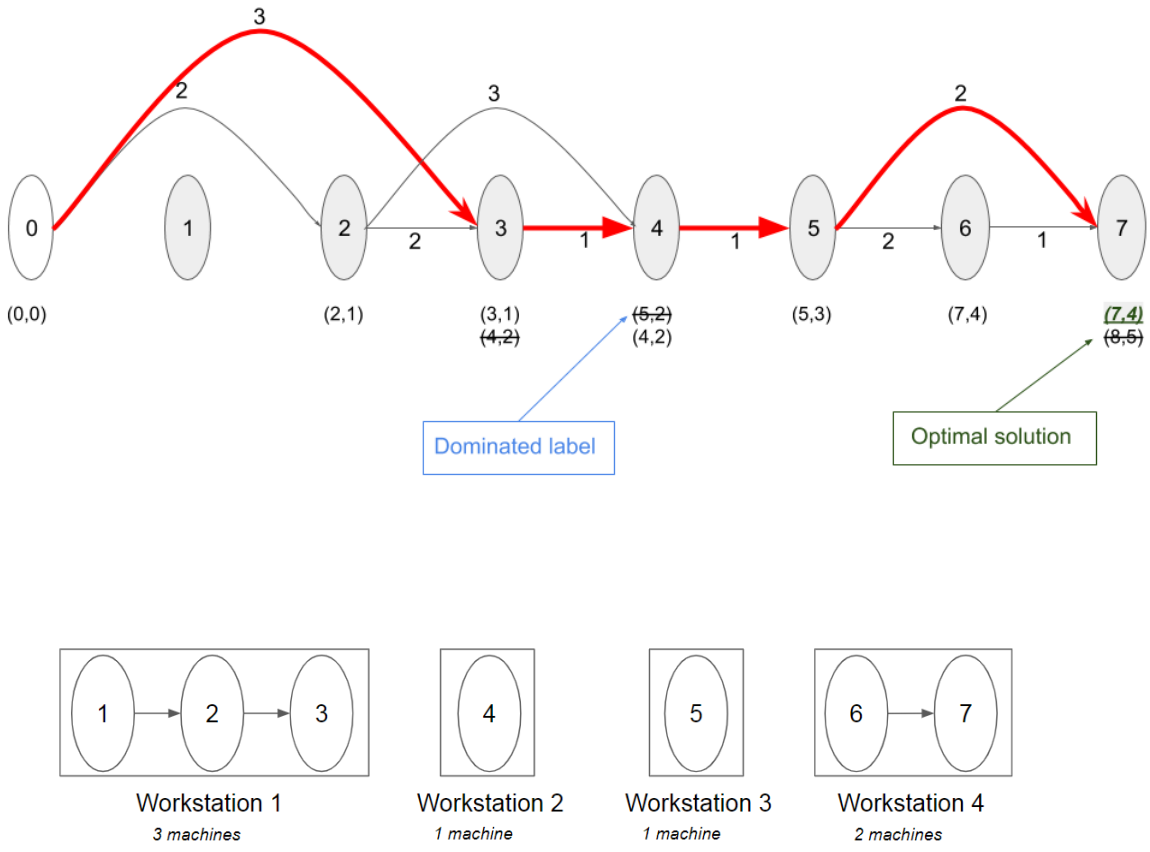

Figure 4. *Trace of the split algorithm and scheme of the optimal solution.*

5. Iterated Local Search

The split can be used to solve the RTLB problem to optimality for a given giant sequence. Using the split procedure like a black box for evaluating a giant sequence allows us to transform the RTLB problem to the search of the best giant sequence. This is the aim of the method proposed in this section.

Once having computed a starting giant sequence, we perform an iterated local search in the space of the giant sequences, the split is used to evaluate **each** giant sequence. The general scheme is depicted in Algorithm 2. **In this algorithm,** $c(S)$ **denotes the cost of solution S, i.e., the number of machines used by** S**.**

Algorithm 2 ILS algorithm hybridised with split

INPUT: An instance of RTLB.
OUTPUT: S^*, best found solution.

1: Compute a compatible giant sequence: σ
2: Perform split: $S = split(\sigma)$
3: Record best known sequence: $(S^*, \sigma^*) = (S, \sigma)$
4: **while** Stopping criterion ILS is not met **do**
5: **while** Stopping criterion LS is not met **do**
6: Choose a random neighbour of σ: σ'
7: Perform split: $S' = split(\sigma')$
8: **if** $c(S') \leq c(S)$ **then**
9: $(S, \sigma) = (S', \sigma')$
10: **end if**
11: **end while**
12: **if** $c(S) \leq c(S^*)$ **then**
13: $(S^*, \sigma^*) = (S, \sigma)$
14: **end if**
15: $(S, \sigma) = Perturbation[(S^*, \sigma^*)]$
16: **end while**

Unlike local search that is trapped in a local optimum, iterated local search (ILS) introduces more diversification by applying a perturbation to the local optimum and iterating the local search. It exploits the idea that performing local search from a pretty good solution (perturbed local optimum) is more effective than starting from a random solution. ILS has proven to be efficient for various combinatorial optimisation problems: [Lourenço, Martin, and Stützle(2010)], [Stützle(2006)], [Dong, Huang, and Chen(2009)].

We start with computing a compatible giant sequence σ (giant sequence corresponding to a feasible solution). Then a local search is performed from σ (lines 5-11): a random neighbour σ' is selected and evaluated thanks to split. σ is updated if $c(S') \leq c(S)$ where S and S' denote the solutions corresponding respectively to σ and σ' (lines 8-10). The local search terminates when the stopping criterion of the local search is met. **We consider a maximum number of iterations as stopping criterion of the local search. The best known solution** S^* **is updated after each local search (lines 12-14).** A perturbation is applied from the **giant sequence yielding the best known solution** σ^* **(line 15)**. The procedure (local search + perturbation) is iterated while the stopping criterion of the ILS is not met. **We consider a maximum**

number of iterations as stopping criterion of the iterated local search. The giant sequence yielding the overall best solution across the ILS is returned by the algorithm. We detail every step of the ILS in the next subsections: the method to build a compatible giant sequence is described in Subsection 5.1, the neighbourhood system is described in Subsection 5.2 and the perturbation operator is described in Subsection 5.3.

5.1. *Computing a compatible giant sequence*

We should notice that given an instance of the RTLB problem, there could exist no feasible solution respecting some **given** giant sequence (for example if **there is no path between the two extremities of the graph associated with the giant sequence**). This leads us to consider the notion of compatible giant sequence.

Definition 5.1. (Compatible giant sequence) A compatible giant sequence is a giant sequence for which there exists a feasible balancing solution respecting the giant sequence.

Given an instance $\mathcal{I}$ of the RTLB problem and a giant sequence σ, σ is compatible with respect to $\mathcal{I}$ if and only if the two conditions are satisfied:

(1) σ respects precedence constraints.
(2) $split(\mathcal{I}, \sigma)$ returns a solution.

If no inclusion, exclusion, accessibility, M_{max}**,** N_{max} **or** s_{max} constraints are taken into consideration, **then** it is easy to compute a compatible giant sequence. Indeed, any giant sequence respecting the precedence constraints is compatible in this case. **Yet**, if we consider **some combination of those** constraints, any sequence respecting the precedence constraints is no longer guarantied to be compatible.

Theorem 5.2. ***Given an instance $\mathcal{I}$ of the RTLB problem, the problem of finding a compatible giant sequence is NP-Hard.***

Proof. **We show that the TSP (Travelling Salesman Decision Problem) is reduced to the considered problem. Indeed, the TSP is the task of deciding whether there exists a "tour" visiting a given set of n_c cities and returning to the initial city such that the total distance travelled is less or equal to a constant B. Then, an instance of the TSP could be mapped with an instance of the RTLB problem where operations represent cities, the setup times between operations represent distances between cities and such that $I = N, E = \emptyset, Pos = \{0\}, Pos_i = \{0\}$ $(\forall i \in N), s_{max} = 1, M_{max} = 1, N_{max} = n_c, d_i = 0$ $(\forall i \in N)$ and $C = B$. This is a polynomial time reduction which justifies the NP-Hardness of our problem.** □

Corollary 5.3. ***The problem of finding a feasible solution for the RTLB problem is NP-Hard.***

Definition 5.4. (Weakly compatible giant sequence) We define a weakly compatible giant sequence as a giant sequence for which there exists a balancing solution that is feasible with relaxing the maximum number of workstations constraint (s_{max}).

In order to compute a compatible giant sequence, we proceed in two steps:

- **Computing a weakly compatible giant sequence.**
- **Repairing the giant sequence to make it compatible.**

The two steps are described below.

5.1.1. Computing a weakly compatible giant sequence

Theorem 5.5. ***Given an instance $\mathcal{I}$ of the RTLB problem, the problem of finding a weakly compatible giant sequence is NP-Hard.***

Proof. **The same reduction as Theorem 5.2. is performed without consideration of s_{max}.** □

Even the problem of determining a weakly compatible giant sequence being NP-Hard, we are content to propose an exponential algorithm (but quite fast in practice) to obtain such a sequence (Algorithm 3). Then, a repair procedure is suggested to obtain a compatible giant sequence from a weakly compatible giant sequence (Algorithm 4). Algorithm 3 works in three steps:

Step 1 Gather the operations according to inclusion constraints: we check easily that the inclusion constraints define an equivalence relation (denoted Inc). Thus, we can compute the quotient set:

$$N/Inc = \{S_1, S_2, \ldots, S_k\}$$

In other words, S_i are groups of operations, built such that if there exists an inclusion relation between two tasks, then they are in the same S_i.

Step 2 **Sequence operations on each set:** optimally sequence the operations in the subsets $S_i, 1 \leq i \leq k$ **while respecting precedence constraints**: we solve an **Asymmetric Travelling Salesman problem** (ATSP) where operations represent cities and setup times represent distances between cities. **This is performed thanks to a dynamic program adapted from [Held and Karp(1962)]. In this way, the subsets S_i become subsequences denoted $\tilde{S}_i$.**

Step 3 Build the weakly compatible giant sequence respecting the precedence constraints.

Algorithm 3 Pseudo-algorithm to compute a weakly compatible giant sequence

---

INPUT: An instance of RTLB problem.
OUTPUT: σ giant sequence that is hopefully weakly compatible.

Step 1 Gather the operations according to inclusion constraints
Step 2 **Sequence operations on each set [Held and Karp(1962)]**
Step 3 Build the weakly compatible giant sequence σ respecting the precedence constraints:
  (a) $\sigma = \emptyset$.
  (b) Select randomly a **subsequence** $\tilde{S}_i$ such that all predecessors of operations in $\tilde{S}_i$ are already contained in σ.
  (c) Append $\tilde{S}_i$ to σ.
  (d) Return to (b) while the number of operations in σ is inferior to n.

Remark 1. Since Algorithm 3 uses the dynamic programming algorithm from [Held and Karp(1962)] in step (2), it is exponential in the size of the subsets $S_i, 1 \leq i \leq k$. However, the size of every subset is bounded by N_{max} which is usually much smaller than the number of operations. In practice, Algorithm 3 is very efficient both in computation time and memory usage.

Definition 5.6. (Triangular inequality) We say that the setup times respect the triangular inequality if:

$$t_{i,j} + t_{j,k} \leq t_{i,k}, \forall (i,j,k) \in N \times N \times N$$

In the following, $Workload(\tilde{S})$ where $\tilde{S}$ is a subsequence of operations defines the sum of the processing times and the setup times induced by the subsequence. For example, $Workload(\text{"}1,2,3\text{"}) = d_1 + t_{1,2} + d_2 + t_{2,3} + d_3 + t_{3,1}$.

Theorem 5.7. *Given a feasible instance $\mathcal{I}$ of the RTLB problem where the setup times respect the triangular inequality, algorithm 3 returns a weakly compatible giant sequence.*

Proof. Let us consider the solution where each **subsequence** $\tilde{S}_i, 1 \leq i \leq k$ is allocated to a different workstation. This solution respects the giant sequence outputted by Algorithm 3. **Besides, it is easy to see that this solution is feasible with respect to inclusion, exclusion, accessibility, precedence and N_{max} constraints. It remains to verify that the M_{max} constraint is satisfied. To do this, we must check that for each $\tilde{S}_i$, $Workload(\tilde{S}_i) \leq C.M_{max}$. Since all the operations of each subsequence $\tilde{S}_i$ are linked with inclusion constraints, they should all be included in the same workstation in any feasible solution. In other words, for each subsequence $\tilde{S}_i$, there must exist a subsequence $\tilde{S}'_i$ such that:**

(1) $\tilde{S}_i \subset \tilde{S}'_i$.
(2) $Workload(\tilde{S}'_i) \leq C.M_{max}$.

Since the triangular inequality is satisfied, (1) implies that $Workload(\tilde{S}_i) \leq Workload(\tilde{S}'_i)$. Therefore, (2) implies that $Workload(\tilde{S}_i) \leq C.M_{max}$. □

If the setup times do not respect the triangular inequality, Algorithm 3 remains a good heuristic for computing a weakly compatible giant sequence.

Remark 2. The solution described in the proof of Theorem 5.7. could be obtained by a modified version of split, we denote it $\widetilde{split}$. In $\widetilde{split}$, line 9 is deleted and line 21 is replaced by:

$$C^* := \begin{cases} Min_{(a_i,b_i) \in L_n; b_i \leq s_{max}}(a_i) & \text{If } Min_{(a_i,b_i) \in L_n}(b_i) \leq s_{max} \\ Min_{(a_i,b_i) \in L_n}(b_i) & \text{Otherwise.} \end{cases}$$

$\widetilde{split}$ allows solutions exceeding s_{max} workstations. If feasible solutions with at most s_{max} workstations exist, $\widetilde{split}$ has the same behaviour as split. Otherwise, it minimises the number of workstations.

5.1.2. Computing a compatible giant sequence from a weakly compatible giant sequence

After obtaining a **weakly compatible giant sequence**, **Algorithm 4** is designed to obtain **compatible giant sequence**. **Applied after Algorithm 3, the procedure gives a method for computing a compatible giant sequence.** The procedure works as follows: **starting** from a weakly compatible giant sequence, a local search with insertion neighbourhood (subsection 5.2) is applied to **obtain** a compatible giant sequence. A neighbour is accepted if its split uses less workstations.

Algorithm 4 Algorithm to compute a compatible giant sequence from a weakly compatible giant sequence

INPUT: σ, a weakly compatible giant sequence and $\mathcal{I}$ an instance of the RTLB problem.
OUTPUT: σ, a (**hopefully**) compatible giant sequence.

while (Stopping condition is not met) And (σ is not compatible) **do**
  Choose a random neighbour of σ: σ'
  if $\widetilde{split}(\mathcal{I}, \sigma')$ returns a solution **then**
    $S' = \widetilde{split}(\mathcal{I}, \sigma')$
    if Number of workstations in $S' \leq$ Number of workstations in S **then**
      $(S, \sigma) = (S', \sigma')$
    end if
  end if
end while

5.2. *The neighbourhood system*

The neighbourhood system described in this subsection is used in Algorithm 2 and Algorithm 4. We use a simple insertion neighbourhood: insert an operation in a different position of the giant sequence. This neighbourhood is applied in such way that the precedence constraints are respected. Given the giant sequence σ:

$$\sigma = (\sigma_1, \ldots, \sigma_n), \text{where } \sigma_i \text{ is the operation at the } i^{th} \text{ position of } \sigma$$

a random neighbour is **obtained** by selecting a random operation $\sigma_i, 1 \leq i \leq n$. Once this operation selected, two operations must be identified $\sigma_{f(i)}$ and $\sigma_{l(i)}$ such that

$$l(i) = max\{j; j < i \text{ and } (\sigma_j, \sigma_i) \in P\}, \text{the position of the last predecessor of } \sigma_i \text{ in } \sigma$$

and

$$f(i) = min\{j; i < j \text{ and } (\sigma_i, \sigma_j) \in P\}, \text{the position of the first successor of } \sigma_i \text{ in } \sigma$$

Then a random position is selected between positions $l(i)$ and $f(i)$ (**uniform selection in** $\{l(i)+1, \ldots, f(i)-1\}$) to (re)insert operation σ_i.

5.3. *Perturbation operator*

The perturbation operator in the iterated local search stands for applying the neighbourhood operator 3 times. Each of the three neighbourhood moves is **repeated** until a compatible giant sequence is found. **In other words, after the application of each operator, we test the compatibility of the resulting giant sequence by applying split: if it is not compatible, we reject it and apply the operator again.**

6. Experimentation

We describe in this section the experimentation held on a computer equipped with 16 Go in RAM and i7-4790 CPU (3.60 GHz). Algorithms were implemented in JAVA 8.

We compare our resolution method with the latest resolution method suggested in literature: a genetic algorithm that uses either an heuristic or a MIP chromosome decoder **to solve the balancing subproblem** [Borisovsky, Delorme, and Dolgui(2013)]. The authors mention on the paper the given cost (**number of machines**) obtained for each instance. The comparison is done by running our method on the same sets of 15 instances.

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: $|Pos| = 7$.

6.1. *Building a compatible giant sequence*

We have described a method to compute a starting compatible giant sequence: it refers to apply Algorithm 3 then Algorithm 4.

Table 4 shows the performance of the method. "Cost of the solution" refers to the number of machines of the solution corresponding to the giant sequence obtained with the initial random seed. The latter is used to initiate the remainder of experiments. The two remaining columns show the mean and standard deviation of the cost over 10 independent runs of the method.

The method can compute a starting compatible giant sequence for all the instances. Besides it converges in less than 1 min for most instances and random seeds.

Table 4. Performance of the method to build an initial solution.

	Method to build an initial compatible giant sequence		
Instance	Cost of the solution	Mean	Standard deviation
A1	37	39.3	1.41
A2	39	38.5	1.68
A3	38	39.1	1.29
A4	38	37.8	1.24
A5	41	39.4	1.74
A6	41	38.9	1.3
A7	41	39.9	1.13
A8	38	38.6	1.2
A9	40	38.5	1.36
A10	41	40.5	0.92
A11	40	38.6	0.79
A12	40	39	0.77
A13	38	40.4	1.56
A14	39	38.8	1.24
A15	39	39.2	0.74

6.2. *Split-based ILS*

Tables 5 and 6 show the results of the split-based ILS with different configurations. ILS(x,y) means x iterated local searches of y iterations each (i.e. y neighbours visited in each local search). So the stopping criterion of the local search is the number of neighbours visited and the stopping criterion of the ILS is the number of local searches. Starting from an initial giant sequence, 10 independent runs of the iterated local search are performed. We use the following notations in the tables:

- min: minimum number of machines obtained by the split-based ILS over 10 independent runs. **This columns allows to compare with the genetic algorithm from the literature, because in this last only the min is reported.**
- max: maximum number of machines obtained by the split-based ILS over 10 independent runs.
- mean: average number of machines obtained by the split-based ILS over 10 independent runs.
- σ: Standard deviation of the cost obtained by the split-based ILS over 10 independent runs.

Table 5 shows 3 configurations of the ILS with 100 iterated local searches. The number of iterations in the local searches varies from 500 to 2000. Clearly, with a fixed number of local searches, increasing the number of iterations in the local search leads to improving the results (min, max, mean and standard deviation decreased).

The idea of table 6 is to investigate the behaviour of the ILS when running for a long time by running ILS(100,10'000) and ILS(1'000,1'000). ILS(100,10'000) is giving the best results when compared with all the configurations, but it requires also a large CPU time: 5000". ILS(100,10'000) outperforms ILS(1'000,1'000) despite ILS(1'000,1'000) taking more time to run: 6000". The table also contains a configuration with small x and y (ILS(50,100)) to investigate the quality of the solution when the number of iterations is low. This configuration runs in 250" and already outperforms the algorithm of the literature.

From Tables 5 and 6, we notice that the standard deviation is very low

Table 5. Split-based ILS with different configurations: ILS(100,500) means 100 iterated local searches of 500 iterations each

	ILS(100,500)				ILS(100,1'000)				ILS(100,2'000)			
Instance	min	max	mean	σ	min	max	mean	σ	min	max	mean	σ
A1	30.0	31.0	30.1	0.3	29.0	30.0	29.5	0.5	**28.0**	29.0	28.9	0.3
A2	**27.0**	29.0	28.1	0.539	**27.0**	28.0	27.7	0.458	**27.0**	28.0	27.2	0.4
A3	28.0	29.0	28.5	0.5	**27.0**	28.0	27.5	0.5	**27.0**	28.0	27.4	0.49
A4	**27.0**	29.0	28.3	0.64	**27.0**	28.0	27.6	0.49	**27.0**	28.0	27.3	0.458
A5	**28.0**	30.0	29.2	0.748	**28.0**	30.0	28.9	0.539	**28.0**	29.0	28.1	0.3
A6	28.0	30.0	29.0	0.632	**27.0**	29.0	28.1	0.539	**27.0**	28.0	27.9	0.3
A7	29.0	31.0	29.9	0.7	29.0	30.0	29.6	0.49	**28.0**	30.0	29.0	0.447
A8	29.0	31.0	29.6	0.663	29.0	30.0	29.6	0.49	**28.0**	30.0	29.0	0.447
A9	28.0	30.0	28.4	0.663	**27.0**	29.0	28.0	0.447	**27.0**	28.0	27.4	0.49
A10	31.0	32.0	31.5	0.5	**30.0**	32.0	31.2	0.6	31.0	32.0	31.1	0.3
A11	28.0	29.0	28.5	0.5	**27.0**	29.0	28.0	0.447	**27.0**	28.0	27.3	0.458
A12	29.0	31.0	29.9	0.7	29.0	30.0	29.5	0.5	**28.0**	30.0	28.8	0.6
A13	30.0	32.0	30.3	0.64	**29.0**	30.0	29.4	0.49	**29.0**	30.0	29.1	0.3
A14	28.0	30.0	29.0	0.447	28.0	29.0	28.5	0.5	**27.0**	28.0	27.9	0.3
A15	**28.0**	29.0	28.7	0.458	**28.0**	30.0	28.8	0.6	**28.0**	29.0	28.1	0.3

Table 6. Split-based ILS with different configurations: ILS(50,500) means 100 iterated local searches of 1'000 iterations each

	ILS(50,500)				ILS(100,10'000)				ILS(1'000,1'000)			
Instance	min	max	mean	σ	min	max	mean	σ	min	max	mean	σ
A1	30.0	31.0	30.3	0.458	**28.0**	29.0	28.2	0.4	**28.0**	30.0	29.0	0.447
A2	27.0	29.0	28.5	0.671	**26.0**	27.0	26.6	0.49	**26.0**	28.0	27.0	0.447
A3	29.0	29.0	29.0	0.0	**26.0**	27.0	26.9	0.3	27.0	28.0	27.3	0.458
A4	28.0	29.0	28.7	0.458	**26.0**	27.0	26.6	0.49	27.0	28.0	27.2	0.4
A5	29.0	31.0	29.8	0.6	**27.0**	28.0	27.4	0.49	**27.0**	29.0	28.1	0.539
A6	29.0	30.0	29.5	0.5	**26.0**	28.0	27.0	0.447	27.0	28.0	27.3	0.458
A7	30.0	32.0	30.8	0.872	**28.0**	29.0	28.3	0.458	**28.0**	30.0	29.0	0.447
A8	29.0	31.0	30.0	0.632	**28.0**	29.0	28.3	0.458	**28.0**	30.0	29.0	0.447
A9	28.0	30.0	28.6	0.663	**27.0**	27.0	27.0	0.0	**27.0**	28.0	27.2	0.4
A10	31.0	33.0	32.1	0.539	**29.0**	30.0	29.9	0.3	30.0	31.0	30.5	0.5
A11	28.0	29.0	28.7	0.458	**26.0**	27.0	26.9	0.3	27.0	28.0	27.1	0.3
A12	30.0	31.0	30.3	0.458	**28.0**	29.0	28.3	0.458	**28.0**	30.0	28.8	0.6
A13	30.0	32.0	30.8	0.748	**28.0**	29.0	28.2	0.4	**28.0**	29.0	28.9	0.3
A14	29.0	30.0	29.4	0.49	**27.0**	28.0	27.3	0.458	**27.0**	28.0	27.5	0.5
A15	28.0	30.0	29.0	0.632	**27.0**	28.0	27.3	0.458	**27.0**	29.0	28.1	0.539

(almost always below 0.5) which demonstrates the robustness of the proposed method.

The results from [Borisovsky, Delorme, and Dolgui(2013)] are obtained by taking the best (minimum) result out of 10 independent runs, each run being limited to 15 minutes. Table 7 compares the results obtained by the split-based ILS (denoted by "our method" in the table) with the results of [Borisovsky, Delorme, and Dolgui(2013)]. In the column denoted "our method", the best result out of 10 independent runs is reported. Three configurations are reported: ILS(50,500) (running in 250"), ILS(100,1'000) (running in 1000") and ILS(100,10'000) (running in 5000"). Despite the genetic algorithms runs in 900", a clear improvement is already observed from 250" of execution time of the split-based ILS.

Table 8 report the CPU time of the different algorithms. We notice that the perturbation takes about 9% of the total computation time of the ILS.

Table 7. Performance of split-based metaheuristic vs literature.

	Genetic algo. of the literature	Our method		
Instance	after 900" of execution time	after 250"	after 1000"	after 5000"
A1	33	30	29	28
A2	33	27	27	26
A3	31	29	27	26
A4	29	28	27	26
A5	32	29	28	27
A6	32	29	27	26
A7	34	30	29	28
A8	31	29	29	28
A9	30	28	27	27
A10	32	31	30	29
A11	30	28	27	26
A12	31	30	29	28
A13	33	30	29	28
A14	31	29	28	27
A15	33	28	28	27

Table 8. Approximate CPU times of the algorithms with different configurations.

Configuration	Average time
ILS(50,500)	250"
ILS(100,500)	450"
ILS(100,1'000)	1000"
ILS(100,2'000)	1300"
ILS(100,10'000)	5000"
ILS(1'000,1'000)	6000"
Genetic algo. of the literature	900"

7. Conclusion and perspectives

We show in this article that balancing a sequence of operations in reconfigurable transfer lines is a polynomial-time problem by modelling it as a constrained shortest path problem in some auxiliary graph. A polynomial algorithm (split) to solve this problem is suggested. We tested the efficiency of split by incorporating it in an iterated local search to solve the RTLB problem.

Computational experiments show that the proposed method drastically improves the results of literature and prove the relevance of split.

Several directions could be taken as a future research:

- **The consideration of other metaheuristics embedding split.**
- **The design of exact methods embedding split.**
- **Studying multiple products by investigating the re-balancing problem taking place when the line should be re-balanced for a new product.**

Acknowledgement

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

[Andrés, Miralles, and Pastor(2008)] Andrés, Carlos, Cristóbal Miralles, and Rafael Pastor. 2008. "Balancing and scheduling tasks in assembly lines with sequence-dependent setup times." *European Journal of Operational Research* 187 (3): 1212 – 1223.

[Battaïa et al.(2012)] Battaïa, Olga, Alexandre Dolgui, Nikolay Guschinsky, and Genrikh Levin. 2012. "A decision support system for design of mass production machining lines composed of stations with rotary or mobile table." *Robotics and Computer-Integrated Manufacturing* 28 (6): 672–680.

[Battaïa and Dolgui(2013)] Battaïa, Olga, and Alexandre Dolgui. 2013. "A taxonomy of line balancing problems and their solution approaches." *International Journal of Production Economics* 142 (2): 259 – 277.

[Beasley(1983)] Beasley, John E. 1983. "Route first—cluster second methods for vehicle routing." *Omega* 11 (4): 403–408.

[Borisovsky, Delorme, and Dolgui(2012)] Borisovsky, Pavel A., Xavier Delorme, and Alexandre Dolgui. 2012. "Balancing reconfigurable machining lines by means of set partitioning model." *IFAC Proceedings Volumes* 45 (6): 426 – 431.

[Borisovsky, Delorme, and Dolgui(2013)] Borisovsky, Pavel A., Xavier Delorme, and Alexandre Dolgui. 2013. "Genetic algorithm for balancing reconfigurable machining lines." *Computers Industrial Engineering* 66 (3): 541 – 547. Special Issue: The International Conferences on Computers and Industrial Engineering (ICCIEs) - series 41.

[Buxey(1974)] Buxey, G. M. 1974. "Assembly Line Balancing with Multiple Stations." *Manage. Sci.* 20 (6): 1010–1021.

[Delorme, Malyutin, and Dolgui(2016)] Delorme, Xavier, Sergey Malyutin, and Alexandre Dolgui. 2016. "A multi-objective approach for design of reconfigurable transfer lines." *IFAC-PapersOnLine* 49 (12): 509 – 514. 8th IFAC Conference on Manufacturing Modelling, Management and Control MIM 2016.

[Dolgui, Guschinsky, and Levin(2000)] Dolgui, A, N Guschinsky, and G Levin. 2000. "Approaches to balancing of transfer lines ith blocks of parallel operations." *Prepr./Inst. of eng. cybernetics of the Nat. acad. of sciences of Belarus* (8): 42.

[Dong, Huang, and Chen(2009)] Dong, Xingye, Houkuan Huang, and Ping Chen. 2009. "An iterated local search algorithm for the permutation flowshop problem with total flowtime criterion." *Computers & Operations Research* 36 (5): 1664–1669.

[Essafi, Delorme, and Dolgui(2010a)] Essafi, Mohamed, Xavier Delorme, and Alexandre Dolgui. 2010a. "Balancing lines with CNC machines: A multi-start ant based heuristic." *CIRP Journal of Manufacturing Science and Technology* 2 (3): 176–182.

[Essafi, Delorme, and Dolgui(2010b)] Essafi, Mohamed, Xavier Delorme, and Alexandre Dolgui. 2010b. "Balancing machining lines: a two-phase heuristic." *Studies in Informatics and Control* 19 (3): 243–252.

[Essafi et al.(2010)] Essafi, Mohamed, Xavier Delorme, Alexandre Dolgui, and Olga Guschinskaya. 2010. "A MIP approach for balancing transfer line with complex industrial constraints." *Computers Industrial Engineering* 58 (3): 393 – 400.

[Held and Karp(1962)] Held, Michael, and Richard M Karp. 1962. "A dynamic programming approach to sequencing problems." *Journal of the Society for Industrial and Applied Mathematics* 10 (1): 196–210.

[Koren and Shpitalni(2010)] Koren, Yoram, and Moshe Shpitalni. 2010. "Design of reconfigurable manufacturing systems." *Journal of Manufacturing Systems* 29 (4): 130 – 141.

[Koren, Wang, and Gu(2017)] Koren, Yoram, Wencai Wang, and Xi Gu. 2017. "Value creation through design for scalability of reconfigurable manufacturing systems." *International Journal of Production Research* 55 (5): 1227–1242.

[Lahrichi et al.(2018)] Lahrichi, Y, L. Deroussi, N. Grangeon, and S. Norre. 2018. "Reconfigurable transfer line balancing problem: A new MIP approach and approximation hybrid algorithm." In *MOSIM 2018 (Modélisation et Simulation)*, Toulouse, France.

[Lourenço, Martin, and Stützle(2010)] Lourenço, Helena R, Olivier C Martin, and Thomas Stützle. 2010. "Iterated local search: Framework and applications." In *Handbook of metaheuristics*, 363–397. Springer.

[Martino and Pastor(2010)] Martino, Luigi, and Rafael Pastor. 2010. "Heuristic procedures for solving the general assembly line balancing problem with setups." *International Journal of Production Research* 48 (6): 1787–1804.

[Mehrabi et al.(2002)] Mehrabi, M. G., A. G. Ulsoy, Y. Koren, and P. Heytler. 2002. "Trends and perspectives in flexible and reconfigurable manufacturing systems." *Journal of Intelligent Manufacturing* 13 (2): 135–146.

[Prins, Lacomme, and Prodhon(2014)] Prins, Christian, Philippe Lacomme, and Caroline Prodhon. 2014. "Order-first split-second methods for vehicle routing problems: A review." *Transportation Research Part C: Emerging Technologies* 40: 179–200.

[Rabbani et al.(2016)] Rabbani, Masoud, Reyhaneh Siadatian, Hamed Farrokhi-Asl, and Neda Manavizadeh. 2016. "Multi-objective optimization algorithms for mixed model assembly line balancing problem with parallel workstations." *Cogent Engineering* 3 (1): 115–203.

[Stützle(2006)] Stützle, Thomas. 2006. "Iterated local search for the quadratic assignment problem." *European Journal of Operational Research* 174 (3): 1519–1539.

[Vilarinho and Simaria(2006)] Vilarinho, P. M., and A. S. Simaria. 2006. "ANTBAL: an ant colony optimization algorithm for balancing mixed-model assembly lines with parallel workstations." *International Journal of Production Research* 44 (2): 291–303.