Investigating two variants of the Sequence-Dependent Robotic Assembly Line Balancing Problem by means of a split-based approach

Youssef Lahrichi, Laurent Deroussi, Nathalie Grangeon, Sylvie Norre

Investigating two variants of the Sequence-Dependent Robotic Assembly Line Balancing Problem by means of a split-based approach

Youssef Lahrichi[a], Laurent Deroussi[b], Nathalie Grangeon[b] and Sylvie Norre[b]

[a]HuManis laboratory of EM Strasbourg Business School, 61 avenue de la Forêt Noire, F-67000, Strasbourg, France; [b]Université Clermont-Auvergne, CNRS, Mines de Saint-Étienne, LIMOS, 63000 Clermont-Ferrand, France

ARTICLE HISTORY
Compiled January 17, 2022

ABSTRACT
The Robotic Assembly Line Balancing Problem (RALBP) is a joint optimization problem that is concerned with assigning both assembly operations and robots to workstations that are placed within a straight line. The objective of RALBP-2 is to minimize the cycle time which is the maximum time spent on a workstation by the product being assembled. Sequence-dependent setup times are considered which raises the problem of sequencing the operations assigned to each workstation. Both the durations of the operations and the setup times depend on the robot. Two different variants are identified from literature. The first variant assumes that, given a set of types of robots, each type of robot can be assigned to multiple workstations without any limitation. Given a set of robots, the second variant forces each robot to be assigned to at most one workstation. Both assumptions are studied in this paper. The particular case of a given giant sequence of operations is solved thanks to a polynomial optimal algorithm. The latter algorithm, called split, is then embedded in a metaheuristic framework that explores the space of giant sequences. Benchmark data sets from literature are considered in the experimental section. A comparative study with other methods from literature shows the competitiveness of the suggested approach.

KEYWORDS
Assembly line balancing; robotic assembly line; sequence-dependent setup times; polynomial case; metaheuristic.

1. Introduction

We have been assisting for years to the growing robotization of production lines. According to the international federation of robotics, a 14% increase is observed each year in terms of operational industrial robot jobs. Robots are replacing increasingly humans in repetitive or hard manufacturing tasks. They offer many benefits in the context of manufacturing:

- Accuracy: the more and more manufacturing tasks require cutting edge precision beyond the reach of a human operator.

Corresponding author: Y. Lahrichi. Email: youssef.lahrichi.contact@gmail.com

- Quality: robots allow better monitoring of quality to help reduce the quantity of waste and inconsistent products.
- Cost: despite higher setup cost, robots give a higher return on investment since their performance is much higher. Besides, robots do not require any training/learning cost.

Assembly lines, which follow this robotization trend, first appeared at the beginning of the 20^{th} century in the factories of Ford Motor. They introduced an important innovation in how to organize the work in a workshop. Indeed, the operations are divided between the operators who are placed along a flow line. The economic interests of assembly lines are considerable, among which, an important increase in productivity. This allowed a rapid development of assembly lines all along the 20^{th} century. These were often used for mass-production of inexpensive products. Even today, with the change in the economic model, assembly lines continue to adapt to new societal and industrial challenges.

Assembly lines are typically composed of a series of workstations connected by a material handling system. The product is processed in each workstation, then is moved to the next workstation thanks to a conveyor. A set of assembly operations is assigned to each workstation. The workload of a workstation designates the sum of the durations of the operations assigned to it. The cycle time represents the largest workload among all workstations. It is the maximum time spent on a workstation by the product being assembled. The cycle time can also be described as the duration separating the exit of two assembled products from the last workstation of the assembly line. The smaller the cycle time is, the more products are processed by the line. For this reason, the cycle time is considered as an indicator of efficiency.

Balancing an assembly line refers to the decision problem of assigning the operations to the workstations. In practice, some constraints must be respected. Precedence constraints are frequently encountered. They link pairs of operations such that one operation must precede another, which means that it must be assigned either to the same workstation or to a workstation placed before on the line. Type II assembly line balancing problems are considered in this paper. They are concerned with minimizing the cycle time with a given maximum number of workstations.

In the context of robotic assembly lines, different robots are available to perform the operations. The durations of the operations depend on the type of robot selected for the workstation. The Robotic Assembly Line Balancing Problem (RALBP) is concerned with simultaneously assigning operations and robots to workstations. RALBP appears to be well justified with regards to the growing robotization of assembly lines under the "Industry 4.0" era. We consider the problem under two different assumptions:

- First variant: a set of types of robots is given and the same type of robots can be selected for more than one workstation without any limitation.
- Second variant: a set of robots is given and each robot can be assigned to at most one workstation.

In the context of the first variant, the decision-maker designs the line based on the types of robots sold by his Original Equipment Manufacturer (OEM). While in the second variant, the decision-maker designs the line based on an existing fleet of robots.

Sequence-dependent setups are encountered in many workshop problems [Allahverdi et al.(2008)]. Setups are necessary to provide for tool change or product handling that can occur between two operations. Sequence-dependent setups in the context of assembly lines were only considered recently in

[Andres, Miralles, and Pastor(2008)]. [Scholl, Boysen, and Fliedner(2013)] cite some interesting situations where sequence-dependent setup times are encountered in the context of assembly lines:

- In many industries where large products are manufactured, the operations are performed on different positions of the workpiece. This may require the operator to move between the positions where the consecutive operations are performed. [Scholl, Boysen, and Fliedner(2013)] reports that this walking distance can go up to 10 meters in the context of automotive industry. Alongside with the walking distances of the operators, time to withdraw a part from a container may be required. Scholl reports that for a major German car manufacturer, this walking and withdrawal time takes about 15% of the cycle time.
- Machine-tools often use a single tool at a time. Setup times depending on the sequence of operations are required to change the tool if two consecutive operations require different tools.
- In situations where operations require the workpiece to be put in a specific position, setup is needed for handling the piece between two consecutive operations requiring different positions.
- In many specific situations, a delay must be considered between two operations (for example after gluing or painting).

These last elements justify that the setup times cannot be neglected. Besides, the setup times very often depend on the sequence of operations.

[Andres, Miralles, and Pastor(2008)] define the Sequence-Dependent Simple Assembly Line Balancing Problem (SDSALBP). SDSALBP raises the decision of sequencing the operations in each workstation in addition to the balancing decision. The two decisions must be addressed jointly.

The present paper deals with sequence-dependent setup times in the context of the robotic assembly line balancing problem, i.e., the Sequence-Dependent Robotic Assembly Line Balancing Problem (SDRALBP-2).

The related literature and the contribution of the paper are presented in the next section. Then, we describe both variants of the problem thanks to an illustrative example. The split-based resolution approach is described for the first variant in the fourth section, then it is adapted for the second variant of the problem in the fifth section. Computational experiments, general conclusions and perspective research directions are given in the last sections.

2. Literature review

The Simple Assembly Line Balancing Problem (SALBP) is a well-established combinatorial optimization problem that was first defined by [Salveson(1955)]. It has been studied for decades under various assumptions and constraints, a comprehensive taxonomy can be found in [Battaïa and Dolgui(2013)]. The Robotic Assembly Line Balancing Problem (RALBP) has been much less extensively studied. It was first introduced by [Rubinovitz, Bukchin, and Lenz(1993)]. Two different variants of the problem are identified in literature:

- In the first variant [V1], we are given a set of types of robots. Each type of robots can be selected for multiple workstations without any limitation: [Rubinovitz, Bukchin, and Lenz(1993)], [Yoosefelahi et al.(2012)],

[Nilakantan et al.(2015)], [Çil, Mete, and Ağpak(2016)], [Borba, Ritt, and Miralles(2018)].

- In the second variant [V2], we are given a set of robots each of which can be assigned to at most one workstation: [Levitin, Rubinovitz, and Shnits(2006)], [Gao et al.(2009)], [Janardhanan et al.(2019)].

The problem is more often studied under its first variant since it is the original problem statement introduced by [Rubinovitz, Bukchin, and Lenz(1993)].

The second variant can be of industrial relevance when the decision-maker has a limited fleet of robots. It is a generalization of the first variant since the set of robots can contain robots of the same type. Both variants are studied in the frame of the paper.

The incompatibility between a robot and an operation can be addressed by considering that the corresponding duration is infinite.

Even if the seminal paper ([Rubinovitz, Bukchin, and Lenz(1993)]) considers minimizing the number of workstations with a given cycle time (RALBP-1), most papers consider the objective of minimizing the cycle time (RALBP-2): [Levitin, Rubinovitz, and Shnits(2006)], [Nilakantan et al.(2015)], [Borba and Ritt(2014)], [Janardhanan et al.(2019)]. Some authors consider the latter objective jointly with the objective of minimizing the cost of the robots ([Yoosefelahi et al.(2012)]) or the number of workstations ([Çil, Mete, and Ağpak(2016)]).

Table 1 summarizes some important papers in RALBP literature and situates the present paper. The objective considered is either the cycle time (C), the number of workstations (# St) or the cost of the robots used. Most papers consider minimizing the cycle time. Sequence-dependent setup times have been rarely considered in the context of robotic assembly lines.

Table 1. Literature review.

Paper	Objective			Sequence-dependent	Variant	
	C	# St	Cost	setup times	V1	V2
[Rubinovitz, Bukchin, and Lenz(1993)]		✓			✓	
[Levitin, Rubinovitz, and Shnits(2006)]	✓					✓
[Gao et al.(2009)]	✓					✓
[Yoosefelahi et al.(2012)]	✓		✓		✓	
[Nilakantan et al.(2015)]	✓				✓	
[Çil, Mete, and Ağpak(2016)]	✓	✓	✓		✓	
[Borba, Ritt, and Miralles(2018)]	✓				✓	
[Janardhanan et al.(2019)]	✓			✓		✓
This paper	✓			✓	✓	✓

The Robotic Assembly Line Balancing Problem is similar to existing problems from the manual-manned assembly line literature where a double assignment of operations and workers to the workstations is encountered. There can be several reasons justifying that the two latter assignments must be tackled jointly. For example, the durations of the operations may depend on the worker: in this particular context, the problem is (mathematically) equivalent to the RALBP. The problem is known as the Assembly Line Worker Assignment and Balancing Problem (ALWABP), it has been defined in [Miralles et al.(2008)]. They consider the balancing problem alongside with the assignment of a set of workers with different capabilities to the workstations. In the context of ALWABP, the worker is a limited resource: each worker can be assigned at most once.

Many authors deal with the RALBP by means of metaheuristics: for example

[Levitin, Rubinovitz, and Shnits(2006)] suggest a genetic algorithm for the RALBP-2. A solution is represented by three vectors: a vector that represents a sequence of operations, a vector that represents the workstations and a vector that represents the robots. However, only the first vector is involved in the genetic algorithm. A heuristic is suggested to build a solution from a sequence of operations (giant sequence). The same encoding-decoding technique is adopted in [Nilakantan et al.(2015)].

To the best of our knowledge, [Janardhanan et al.(2019)] is the only paper from literature that deals with the robotic assembly line balancing problem with sequence-dependent setup times. The authors consider the problem only under the second variant. Metaheuristic algorithms are applied to minimize the cycle time. The authors use a permutation of operations and a permutation of robots to encode a solution. To decode those two permutations and build a solution, the authors use a procedure of exponential-time complexity. This bad complexity is due to the fact that all values of cycle time are tested in the worst case.

Contribution of this paper

The paper presents the following novelties:

- For the first variant of the problem, a polynomial case (fixed giant sequence) is solved thanks to a split algorithm. To the best of our knowledge, this particular case has not been solved yet in literature, not even for the problem without setup times. Split is then integrated in a metaheuristic. On instances without setup times, the experiments show that the split-based approach can find optimal solutions and clearly outperforms a method from literature [Nilakantan et al.(2015)].
- For the second variant of the problem, the split algorithm is adapted to decode the permutations of operations and robots that encode a solution. This polynomial-time algorithm stands out from the procedure of [Janardhanan et al.(2019)] that is of exponential-time complexity. The experiments show that the split-based approach outperforms the method from [Janardhanan et al.(2019)].

We have previously used the split algorithm to solve a different balancing problem that considers parallel workstations and the minimization of the number of machines in [Lahrichi et al.(2021)].

In the conference paper [Lahrichi et al.(2020)], we have succinctly described the method for the first variant of the considered problem. In the present paper, the approach is presented in more details. Many improvements on the method have been integrated. Besides, the second variant of the problem is also studied which allows to compare with [Janardhanan et al.(2019)]. More experimental results are presented.

3. Problem statement

In this section, the first and the second variant of the problem are explained thanks to illustrative examples.

3.1. *First variant*

The Robotic Assembly Line Balancing Problem (RALBP) is concerned with simultaneously assigning a set of operations to a set of workstations placed along a serial

assembly line and assigning to each workstation a robot type. The duration of an operation depends on the type of robot used. Sequence-dependent setup times $t^r_{i,j}$ should be considered if operation i is performed just before operation j in some workstation equipped by a robot of type r. We note that the setup times depend on the robot type and raise an additional decision: the sequencing decision that consists of sequencing the operations in each workstation. The first variant of the Sequence-Dependent Robotic Assembly Line Balancing Problem (SDRALBP-2) can be stated as follows:

- **Data:**
  - The set of operations, the set of workstations and the set of types of robots.
  - Precedence relations.
  - A maximum number of workstations.
  - The durations and the setup times.
- **Decisions:**
  - Balancing decision: Assign the operations to the workstations.
  - Selection decision: Select a type of robot for each workstation.
  - Sequencing decision: Sequence the operations in each workstation.
- **Constraints:**
  - Precedence constraints.
  - The maximum number of workstations constraint.
- **Objective:**
  - Minimize the cycle time C which is the maximum workload among the workstations.

The notations used in the remainder of the paper are introduced in Table 2.

Table 2. Notations.

n	Number of operations
N	Set of operations: $\{1, 2, \ldots, n\}$
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
C	Cycle time
n_r	The number of robot categories or types
R	The set of robot types: $\{1, 2, \ldots, n_r\}$
d^r_i	The duration of operation i when it is performed on a workstation equipped with robot type r
$t^r_{i,j}$	The setup time between operations i and j when i is performed just before j on a workstation equipped with robot type r

We give a numerical example for the first variant of the problem. It will be used in the remainder of the paper. A product needs 8 operations in order to be assembled. Three types of robots are considered. The precedence graph, the durations and the setup times are given respectively in Figure 1, Table 3 and Table 4. A maximum number of stations is given: $s_{max} = 3$.

The solution depicted in Figure 2 represents a feasible solution with three workstations. Operations 2, 1 and 3 are assigned to workstation 1, operations 4 and 5 are assigned to workstation 2 and operations 7, 6 and 8 are assigned to workstation 3. A robot of type 2 is used in the first and the third workstation. A robot of type 1 is used in the second workstation.

The workloads are calculated as follows:

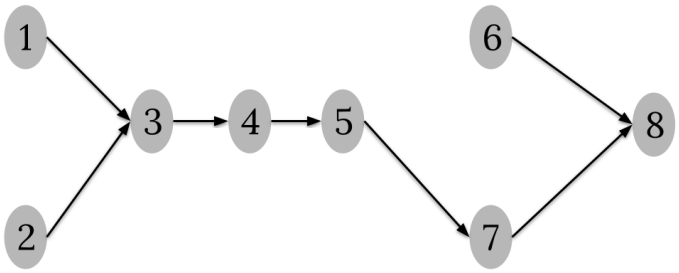

Figure 1. Caption: Precedence graph. Alt text: A precedence graph with eight operations.

Table 3. Durations.

Operation \ Robot type	1	2	3
1	10	15	45
2	15	12	18
3	12	15	30
4	25	20	75
5	30	25	90
6	70	10	15
7	43	65	19
8	36	74	25

Table 4. Sequence-dependent setup times.

$t^1_{i,j}$	1	2	3	4	5	6	7	8
1	-	7	4	1	7	1	7	5
2	4	-	2	4	2	6	8	5
3	9	5	-	3	2	7	9	4
4	4	4	3	-	2	8	9	4
5	1	8	6	1	-	9	4	3
6	9	5	2	7	4	-	3	6
7	6	8	4	2	6	3	-	7
8	5	3	4	7	4	2	9	-

(Robot 1)

$t^2_{i,j}$	1	2	3	4	5	6	7	8
1	-	4	2	8	2	8	4	6
2	9	-	5	3	6	4	8	1
3	8	9	-	1	4	5	5	3
4	6	5	2	-	9	2	8	3
5	4	9	6	7	-	8	4	3
6	5	6	8	7	3	-	3	7
7	3	2	7	6	4	2	-	2
8	4	4	2	9	6	4	6	-

(Robot 2)

$t^3_{i,j}$	1	2	3	4	5	6	7	8
1	-	9	1	8	9	2	3	6
2	1	-	3	8	2	7	9	8
3	3	9	-	2	7	9	7	1
4	1	9	4	-	7	9	3	7
5	1	9	6	5	-	3	2	2
6	4	6	7	3	8	-	9	2
7	3	1	9	1	6	2	-	7
8	7	8	6	9	2	7	3	-

(Robot 3)

- Workload of workstation 1: $W_1 = d^2_2 + t^2_{2,1} + d^2_1 + t^2_{1,3} + d^2_3 + t^2_{3,2} = 62$.
- Workload of workstation 2: $W_2 = d^1_4 + t^1_{4,5} + d^1_5 + t^1_{5,4} = 58$.
- Workload of workstation 3: $W_3 = d^2_7 + t^2_{7,6} + d^2_6 + t^2_{6,8} + d^2_8 + t^2_{8,7} = 164$.

The cycle time of the solution is $C = Max\{W_1, W_2, W_3\} = 164$.

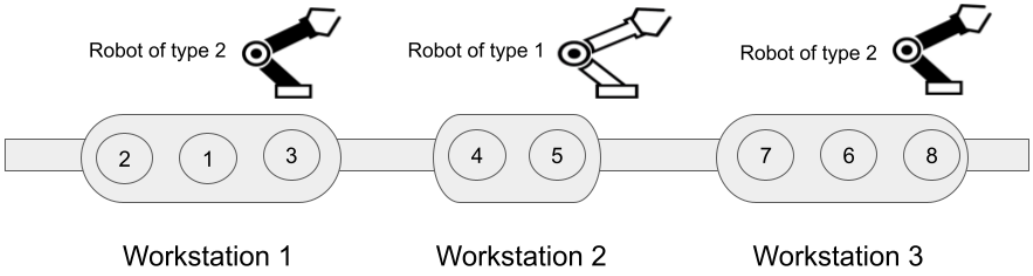

Figure 2. Caption: A feasible solution for the first variant. Alt text: An assembly line with 3 stations. A robot of type 2 is used in the first and third stations, a robot of type 1 is used in the second station.

In workstation 2, the assigned operations are 4 and 5. The best choice for workstation 2 under this assignment of operations is robot type 1. Indeed, even if robot type 2 gives the best durations for operations 4 and 5, the small setup-times on robot type 1 catch up with this gap.

3.2. *Second variant*

With regards to the second variant of the problem, instead of having a set of types of robots each of which can be assigned to multiple workstations, we now have a set of robots each of which can be assigned to at most one workstation. In this context, the input slightly changes: the set of robot types is replaced by the set of robots. We keep the same notations R and n_r respectively for the set of robots and the number of robots. Because it is now assumed that each robot can be assigned to at most one workstation, the solution depicted in Figure 2 is no longer feasible. We give a feasible solution for this second assumption in Figure 3.

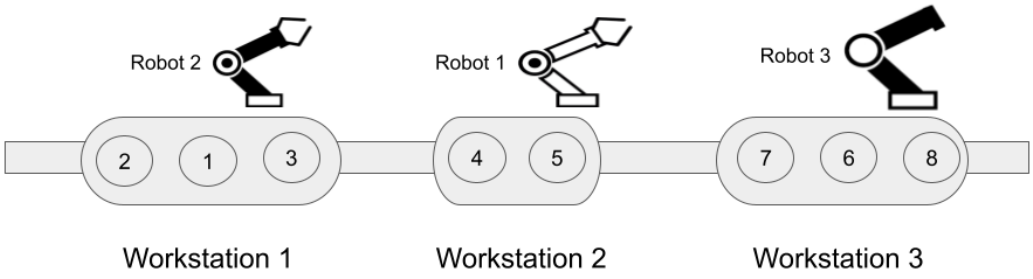

Figure 3. Caption: A feasible solution for the second variant. Alt text: An assembly line with 3 stations. Robot 2, robot 1 and robot 3 are used respectively in the first, the second and the third stations.

4. A split-based resolution approach for the first variant of the problem

For many balancing problems, for example [Levitin, Rubinovitz, and Shnits(2006)] and [Nilakantan et al.(2015)], a particular case is identified: the case where an overall sequence of operations is imposed.

Next subsection is intended to solve this particular case of a given sequence of operations thanks to a novel polynomial algorithm called split. In the second subsection, split is embedded in a metaheuristic.

Split is inspired to us from an homonym method developed by [Beasley(1983)] for the Vehicle Routing Problem (VRP). The VRP is concerned with assigning a set of costumers to a set of vehicles initially located within a depot. The objective is to minimize the total distance travelled by the vehicles. [Beasley(1983)] considers a fixed "giant tour" of costumers. He proves that finding an optimal solution respecting a given giant tour is polynomial. It is equivalent to a shortest path problem in some auxiliary graph. [Prins(2004)] has then used split as a decoding procedure in a metaheuristic which has led to the best-known VRP method for many years. Similar researches were conducted afterwards: [Prins, Lacomme, and Prodhon(2014)]. Split is efficient in a Sequence-First Cluster-Second approach, because it allows to optimally determine the clusters of costumers, given an initial giant tour. We propose here an original way to adapt ideas from VRP to the Sequence-Dependent Robotic Assembly Line Balancing Problem-2 (SDRALBP-2).

Split relies on an auxiliary graph and a procedure to compute the optimal path. Before presenting the general scheme of split, we give some definitions.

Definition 4.1. *(Giant sequence)* Given an instance with n operations, a giant sequence is a permutation of all the operations: $\sigma = (\sigma_1, \sigma_2, \ldots, \sigma_n)$ where σ_i is the operation at position i of the sequence.

Definition 4.2. *(A solution satisfying a giant sequence)* A solution X is said to satisfy a giant sequence $\sigma = (\sigma_1, \sigma_2, \ldots, \sigma_n)$ if for all σ_i, σ_j such that $i < j$ either σ_i and σ_j are assigned to the same workstation in X or the workstation to which σ_i is assigned, is situated before the workstation to which σ_j is assigned. When σ_i and σ_j are assigned to the same workstation, σ_i must be sequenced before σ_j. In other words, the operations must be assigned to the workstations and sequenced while respecting the order given by σ.

Definition 4.3. *(Compatible giant sequence)* We say that σ is a compatible giant sequence with respect to an instance if there exists at least one feasible solution satisfying σ.

Many solutions satisfying a given giant sequence could exist. More precisely, there are an exponential number of solutions respecting a given giant sequence. It corresponds to the different possible ways to cut a giant sequence: $\binom{n-1}{s_{max}-1}$. Split gives an optimal solution respecting σ in polynomial time.

4.1. *A polynomial case: fixed sequence of operations*

Given a giant sequence σ and an instance $\mathcal{I}$, the optimal solution satisfying σ can be obtained thanks to the search of an optimal path in an appropriate auxiliary graph denoted $\mathcal{H}_{\mathcal{I}}(\sigma)$. We describe next the construction of the auxiliary graph $\mathcal{H}_{\mathcal{I}}(\sigma)$.

In the remainder of the subsection, we suppose that the giant sequence is fixed, it is denoted: $\sigma = (\sigma_1, \sigma_2, \ldots, \sigma_n)$.

4.1.1. Auxiliary graph

The graph $\mathcal{H}_{\mathcal{I}}(\sigma) = (V, A)$ is a directed graph, composed of a set of nodes V and a set of arcs A. The set of nodes V is composed of the set of operations N to which we add an additional node that can be seen as a fictitious operation $\sigma_0 = 0$: $V = N \cup \{0\}$. The set of arcs A is composed of all arcs (σ_i, σ_j) such that $i < j$. An arc (σ_i, σ_j) can be interpreted as a workstation to which the subsequence $(\sigma_{i+1}, \sigma_{i+2}, \ldots, \sigma_j)$ is assigned. Besides, $\mathcal{H}_{\mathcal{I}}(\sigma)$ is weighted. The weight of the arc (σ_i, σ_j) is given by:

$$c_{\sigma_i,\sigma_j} = Min_{r \in R} \{ \sum_{k=i+1}^{j} d^r_{\sigma_k} + \sum_{k=i+1}^{j-1} t^r_{\sigma_k,\sigma_{k+1}} + t^r_{\sigma_j,\sigma_{i+1}} \}$$

c_{σ_i,σ_j} indicates the time that is needed to process the subsequence of operations: $(\sigma_{i+1}, \sigma_{i+2}, \ldots, \sigma_j)$. We point out that we could take any robot type that minimizes the workload of the considered workstation.

Theorem 4.4. *The auxiliary graph can be constructed within* $O(n^3.n_r)$.

Proof. The time complexity for constructing the auxiliary graph is equivalent to the complexity for computing the weight $c_{i,j}$ for each arc $(i, j) \in A$. The sum $\sum_{k=i+1}^{j} d^r_{\sigma_k} + \sum_{k=i+1}^{j-1} t^r_{\sigma_k,\sigma_{k+1}} + t^r_{\sigma_j,\sigma_{i+1}}$ can be computed within $O(n)$. It must be computed for each robot. This rises up the complexity to $O(n.n_r)$. The number of arcs is bounded by n^2 which gives the complexity: $O(n.n_r.n^2) = O(n^3.n_r)$. □

4.1.2. A constrained min-max path

In the graph thus constructed, a path from the node 0 to the node σ_n corresponds to a solution of the problem.

Indeed, a path $\{(0, \sigma_{i_1}), (\sigma_{i_1}, \sigma_{i_2}), (\sigma_{i_2}, \sigma_{i_3}), \ldots, (\sigma_{i_{k-1}}, \sigma_{i_k}), (\sigma_{i_k}, \sigma_n)\}$ corresponds to the solution where the subsequence $(\sigma_1, \sigma_2, \ldots, \sigma_{i_1})$ is assigned to the first workstation, $(\sigma_{1+i_1}, \ldots, \sigma_{i_2})$ is assigned to the second workstation,... and $(\sigma_{1+i_k}, \ldots, \sigma_n)$ is assigned to the last workstation. More precisely, there is a bijection between the space of feasible solutions satisfying the giant sequence σ and the paths from 0 to σ_n in $\mathcal{H}_{\mathcal{I}}(\sigma)$. Besides, the objective value of a solution (the cycle time) matches the maximum weight of an arc in the corresponding path in $\mathcal{H}_{\mathcal{I}}(\sigma)$.

To obtain an optimal solution with respect to a giant sequence σ, we can compute a path from 0 to σ_n in $\mathcal{H}_{\mathcal{I}}(\sigma)$ that minimizes the maximum weight of an arc and that uses no more than s_{max} arcs.

The problem of finding a path, between a given pair of vertices, that minimizes the maximum weight of an arc, is known in graph theory as the *min-max path* problem (also known as the *bottleneck path* [Chechik et al.(2016)]).

This problem is polynomial since a shortest path algorithm can be adapted to compute a min-max path. We are dealing with a constrained min-max path because the path should not exceed s_{max} arcs.

Algorithm 1 is proposed to compute the constrained min-max path in $\mathcal{H}_{\mathcal{I}}(\sigma)$. The algorithm is based on labels that keep track of the cycle time and the number of arcs. A label $l = (a, b)$ represents a partial solution, a and b denote respectively the cycle time and the number of workstations used so far by the partial solution. A list L_i of labels is associated to the node σ_i. A label (a, b) dominates a label (a', b') if $a \leq a'$ and

$b \leq b'$ (weak domination).

The propagation of label (a_t, b_t) through the arc (σ_t, σ_i) gives the label $(a_i, b_i) = (Max\{a_t, c_{\sigma_t,\sigma_i}\}, b_t + 1)$: the maximum between a_t and $c_{t,i}$ gives the cycle time, the number of workstations b_t is incremented by 1.

Algorithm 1 Split for the first variant of SDRALBP-2

INPUT ($\mathcal{I}$,σ) where $\mathcal{I}$ is an instance of SDRALBP-2 and σ is a compatible giant sequence.
OUTPUT X: An optimal solution (with the minimal cycle time C^*) respecting σ if there exists a feasible solution

```
Build the graph H_I(σ)
L_0 := {(0,0)}
for t = 1 to n do
    L_t := ∅
end for
for t = 0 to n − 1 do
    for i = t + 1 to n (Propagate labels from L_t) do
        for all (a_t, b_t) ∈ L_t do
            if (b_t < s_max − 1 or i = n) then
                (a_i, b_i) := (Max{a_t, c_{σ_t,σ_i}}, b_t + 1)
                if (a_i, b_i) is not dominated by any label belonging to L_i then
                    L_i := L_i ∪ {(a_i, b_i)}
                    if (a_i, b_i) dominates some element (a'_i, b'_i) ∈ L_i then
                        L_i := L_i \ {(a'_i, b'_i)}
                    end if
                end if
            end if
        end for
    end for
end for
if L_n ≠ ∅ then
    C* := Min_{(a_i,b_i)∈L_n}(a_i)
    Decode the path of cost C* to build X
end if
return X
```

Theorem 4.5. *The algorithm runs in $O(n^4)$.*

Proof. The dominance rule ensures that $|L_i| \leq s_{max}$. The algorithm performs dominance tests for each 3-tuple (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 $m \leq \sum_{k=1}^{n} k = \frac{n(n+1)}{2}$, split runs in $O(n^4)$. □

4.1.3. Illustrative example

We consider the giant sequence $\sigma = (1, 2, 3, 4, 5, 6, 7, 8)$ for the instance described in Subsection 3.1. Split is applied to compute the optimal solution satisfying σ. The auxiliary graph is represented in Figure 4.

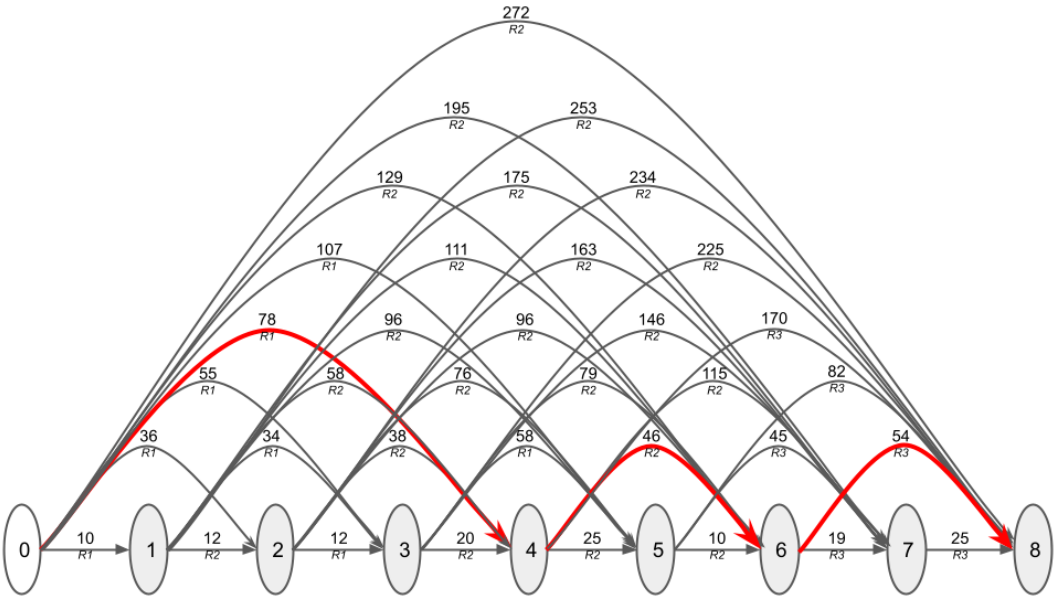

Figure 4. Caption: Auxiliary graph. Alt text: A directed acyclic graph with nine nodes. The graph is weighted and the robot types are written below the weights. The path (0,4),(4,6),(6,8) is highlighted in red.

An optimal path, i.e. a min-max path with at most three arcs, is highlighted in red in Figure 4. The optimal path corresponds to an optimal solution with a cycle time of 78. It is represented in Figure 5.

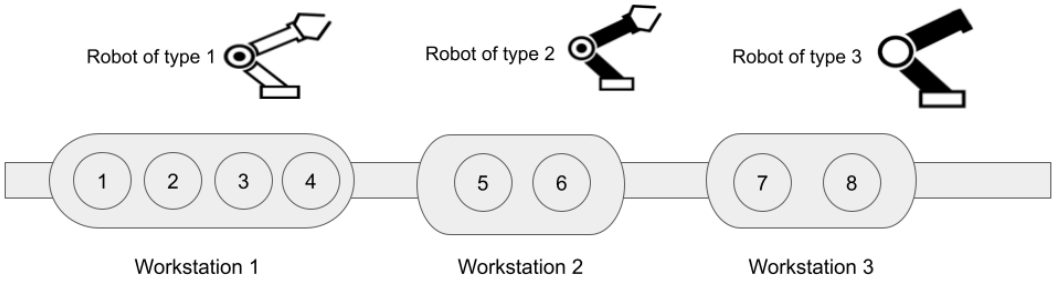

Figure 5. Caption: Optimal solution satisfying the sequence $\sigma = (0, 1, 2, 3, 4, 5, 6, 7, 8)$. Alt text: An assembly line with 3 stations. Robots of type 1, 2 and 3 are used in the first, the second and the third stations.

4.2. *An approach of type Sequence-First Balance-And-Select-Second*

In this subsection, split is used as a decoding procedure in a metaheuristic. A solution is encoded as a giant sequence and split is used to decode it. This approach can be described to be of type "Sequence-First Balance-And-Select-Second". Indeed, the sequencing of operations is first performed by computing a giant sequence. Then, the balancing and the selection decisions are optimally solved by applying split to the giant sequence. The space of giant sequences is explored using a metaheuristic.

This split-based encoding-decoding scheme significantly reduces the search space. We denote it by $[\sigma, split]$. Split acts as an aggressive and efficient guidance technique which allows, for a given sequence, to solve in polynomial time all the other decision subproblems.

Theorem 4.6. *The encoding-decoding technique* $[\sigma, split]$ *preserves an optimal solution for the robotic assembly line balancing problem.*

Proof. Any optimal solution X can be represented by a giant sequence σ_X. The execution of split on σ_X yields either X or any other solution with equivalent objective value since split returns an optimal solution corresponding to a giant sequence. □

The previous theorem shows that the set of optimal solutions has non-empty intersection with the set of solutions given by split (Figure 6).

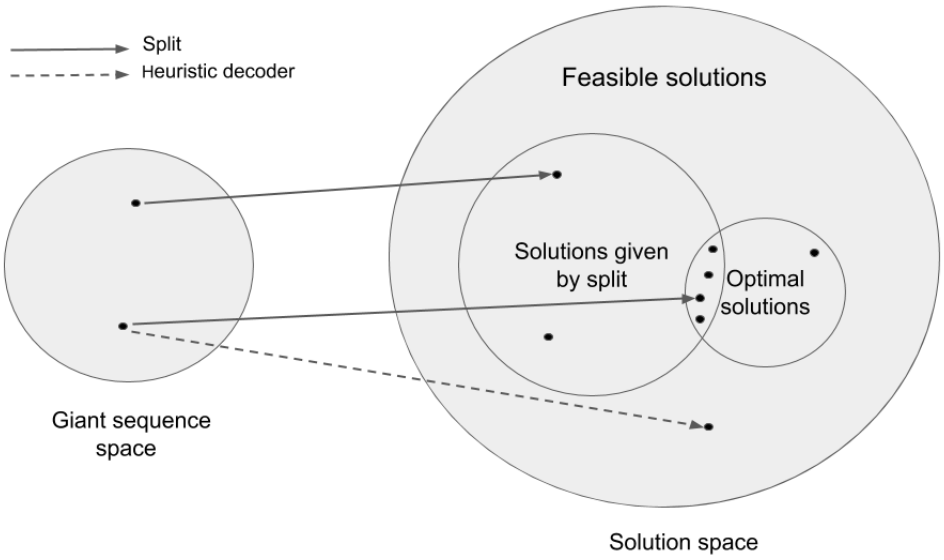

Figure 6. Caption: Split versus heuristic decoder. Alt text: The space of giant sequences is represented smaller next to the space of feasible solutions. Split maps a giant sequence to an optimal solution while an heuristic decoder maps the same giant solution to a sub-optimal solution.

In Figure 6, the space of giant sequences is voluntarily represented smaller than the space of solutions. Split gives an optimal solution when the giant sequence encodes an optimal solution. A heuristic decoder does not guarantee the same. All what remains is searching for a "good" giant sequence by means of metaheuristics.

To demonstrate the superiority of the encoding-decoding technique, we choose a quite simple metaheuristic framework and compare it with more sophisticated metaheuristics from literature that use a different encoding-decoding technique in the experiments section. An iterated local search is used. This metaheuristic has been introduced in [Dong, Huang, and Chen(2009)]. It has been proven to be efficient for recent balancing problems [Abdous(2019)], [Lahrichi et al.(2021)]. The pseudo-algorithm of the split-based iterated local search is given in Algorithm 2.

An initial compatible giant sequence σ is computed first (line 1), the corresponding (optimal) solution is given by split (line 2). A local search is performed with a given maximum number of visited neighbors (LS-MAX). A neighboring giant sequence σ' of σ is then computed (line 7). The corresponding (optimal) solution is given by split (line 8). Each time a neighbor with (strictly) better objective value is reached, the counter is set to 0 (line 12). When a local search terminates, the best recorded solution is perturbed by some perturbation move and the obtained solution is used as a starting solution for the next local search (lines 20-21). A given number of iterated local searches is performed (ILS-MAX).

Algorithm 2 Split-based metaheuristic for the first variant of SDRALBP-2

INPUT An instance of SDRALBP-2, ILS-MAX, LS-MAX.
OUTPUT X^*: A local-optimal solution.

1: Compute an initial compatible giant sequence: σ
2: Build a solution X from σ thanks to split: $X := Split[\sigma]$
3: $[X^*, \sigma^*] := [X, \sigma]$
4: **for** $k = 1$ **to** ILS-MAX **do**
5: Iter := 1
6: **while** Iter $\leq$ LS-MAX **do**
7: Compute a neighboring giant sequence: $\sigma' := Insertion[\sigma]$
8: Build a solution from σ' thanks to split: $X' := Split[\sigma']$
9: **if** Cycle time of $X' \leq$ Cycle time of X **then**
10: $[X, \sigma] := [X', \sigma']$
11: **if** Cycle time of $X' <$ Cycle time of X **then**
12: Iter := 0
13: **end if**
14: **end if**
15: Iter := Iter + 1
16: **end while**
17: **if** Cycle time of $X \leq$ Cycle time of X^* **then**
18: $[X^*, \sigma^*] := [X, \sigma]$
19: **end if**
20: Apply perturbation: $\sigma := Perturbation[\sigma^*]$
21: Update the solution: $X := Split[\sigma]$
22: **end for**
23: **return** X^*

Algorithm 2 relies on a procedure that computes an initial compatible giant sequence (line 1). Any giant sequence respecting precedence constraints is compatible since the solution with a single workstation containing all the giant sequence is feasible. A giant sequence respecting precedence constraints is computed greedily by starting with a random operation without predecessors then by taking at each step a random operation whose predecessors have already been included in the sequence.

The neighborhood move (line 7) stands for an insertion move where an operation is randomly selected and inserted in a different position. The latter move is applied in such a way that precedence constraints are respected.

The perturbation move stands for applying the neighborhood move three times.

During the local search, a new auxiliary graph must be built for each new visited neighbor. To accelerate the algorithm, only parts of the graph that are subject to change are reprocessed. Since we use an insertion move, when an operation at position i is moved to some position j:

- If $j > i$: Arcs (σ_a, σ_b) such that $(b \leq i - 1)$ or $(a \geq j)$ or $(a \geq i$ and $b \leq j - 1)$ remain unchanged.
- If $i > j$: Arcs (σ_a, σ_b) such that $(b \leq j - 1)$ or $(a \geq i)$ or $(a \geq j$ and $b \leq i - 1)$ remain unchanged.

To accelerate the split procedure during the local search, some labels are pruned if their cycle times exceed the best-known cycle time. We note that this speeding-

up technique preserves the optimality of split since all the labels deleted during the execution of split cannot yield any optimal solution.

5. Adaptation of the split-based approach for the second variant of the problem

To deal with the second variant of the problem, the main issue is to ensure that each robot is assigned at most once. The split algorithm is not directly applicable since it can assign the same robot to more than one workstation.

To adapt split not to use the same robot in more than one workstation, we use a permutation of robots alongside with the giant sequence of operations to encode a solution. The permutation does not contain the same robot more than once. During the execution of split, the robots are assigned in the order given by the permutation of robots.

Next subsection is devoted to the description of split for the second variant of the problem. In the second subsection, we explain how it is embedded in a metaheuristic.

5.1. *A polynomial case: fixed sequence of operations and fixed permutation of robots*

We suppose that we have, alongside with a fixed giant sequence of operations $\sigma = (\sigma_1, \sigma_2, \ldots, \sigma_n)$, a fixed permutation of robots $\pi = (\pi_1, \pi_2, \ldots, \pi_{s_{max}})$. We compute next the optimal solution that respects σ such that the k^{th} workstation uses the robot at position k in π, i.e., π_k.

5.1.1. Auxiliary multi-graph

$\mathcal{H}_{\mathcal{I}}(\sigma) = (V, A)$ is a directed weighted multi-graph defined by its set of vertices V and its multi-set[1] of arcs A. The set of vertices does not change from previously: $V = N \cup \{\sigma_0 = 0\}$.

As it is a multi-graph, the same arc (same source and same destination) can be present more than once in the graph. The set of arcs A contains n_r occurrences of the arcs (σ_i, σ_j) such that $i < j$. The occurrences of an arc (σ_i, σ_j) are labelled from 1 to n_r. We denote $(\sigma_i, \sigma_j)_r \in A$ the occurrence number r of arc (σ_i, σ_j), i.e., the occurrence representing the robot r.

The arc $(\sigma_i, \sigma_j)_r$ represents a workstation equipped with robot r to which the operations from σ_{i+1} to σ_j are assigned. The weight of an arc $(\sigma_i, \sigma_j)_r$ is given by:

$$w((\sigma_i, \sigma_j)_r) = \sum_{k=i+1}^{j} d^r_{\sigma_k} + \sum_{k=i+1}^{j-1} t^r_{\sigma_k, \sigma_{k+1}} + t^r_{\sigma_j, \sigma_{i+1}}$$

$w((\sigma_i, \sigma_j)_r)$ represents the time required to perform the subsequence $(\sigma_{i+1}, \ldots, \sigma_j)$ on robot r.

Theorem 5.1. *The auxiliary multi-graph can be constructed within $O(n^3.n_r)$.*

[1] In a multi-set, an element can be present more than once, the multiplicity defines the number of copies of an element in the set

Proof. The sum $\sum_{k=i+1}^{j} d^r_{\sigma_k} + \sum_{k=i+1}^{j-1} t^r_{\sigma_k,\sigma_{k+1}} + t^r_{\sigma_j,\sigma_{i+1}}$ can be computed within $O(n)$. It must be computed for each arc $(i,j)_r$. The number of arcs is bounded by $n_r.n^2$ which gives the complexity: $O(n.n_r.n^2) = O(n^3.n_r)$. □

5.1.2. A constrained min-max path

An optimal solution respecting the giant sequence of operations σ and the permutation of robots π can be found by computing a path from σ_0 to σ_n in $\mathcal{H}_{\mathcal{I}}(\sigma)$ that minimizes the maximum weight of an arc. The path is constrained not to exceed s_{max} arcs and to respect the permutation π, i.e., it must be ensured that the k^{th} arc of the path uses the robot π_k for $1 \leq k \leq s_{max}$. Algorithm 1 can be adapted to compute such a path. Line 10 must be replaced by the two following lines:

$$r := \pi_{b_t+1}$$

$$(a_i, b_i) := (Max\{a_t, w((\sigma_t, \sigma_i)_r)\}, b_t + 1)$$

The first instruction is intended to determine the robot to be used according to π. The second instruction corresponds to label propagation. It is intended to compute the workload of the next workstation. The time complexity of the split algorithm does not change.

5.2. *An approach of type Sequence-And-Select-First Balance-Second*

In this subsection, split is used as a decoding procedure in a metaheuristic. A solution is encoded by a giant sequence of operations σ and a permutation of robots π. Split is used to decode (σ, π). Thus, the approach could be described to be of type "Sequence-And-Select-First Balance-Second". Indeed, the sequencing of operations and the selection of robots are first performed by computing respectively σ and π. Then, the balancing decision is optimally solved by applying split to (σ, π).

We adapt the metaheuristic from the previous section. The local search on the space of giant sequences of operations (with a fixed permutation of robots) is alternated with a local search on the permutations of robots (with a fixed giant sequence of operations). This latter local search is applied each time a neighbor with better or equivalent objective value is obtained. The pseudo-algorithm is described in Algorithm 3.

Algorithm 3 relies on a local search on the permutations of robots with a fixed giant sequence of operations (lines 13, 15). It is described in Algorithm 4. Algorithm 3 relies on a procedure that computes an initial compatible giant sequence and an initial permutation of robots: (σ, π) (line 1). The giant sequence of operations is computed by generating a random giant sequence of operations σ respecting precedence constraints. The initial permutation of robots could be computed by taking any random permutation of robots. However, we propose A heuristic to compute a permutation of robots yielding good solutions:

(1) Compute a solution for the first variant of the problem by applying Algorithm 1 on σ. This solution may not be admissible for the second variant of the problem since it may assign the same robot to multiple workstations.
(2) **While** there exists a robot r assigned to more than one workstation in X

Algorithm 3 Split-based metaheuristic for the second variant of SDRALBP-2

INPUT An instance of SDRALBP-2, ILS-MAX, LS-MAX, Max-Neighbors-1, Max-Neighbors-2.
OUTPUT X^*: A local-optimal solution.

1: Compute an initial compatible giant sequence and an initial permutation of robots: (σ, π)
2: Build a solution X from (σ, π) thanks to split: $X := Split[\sigma, \pi]$
3: $[X^*, \sigma^*, \pi^*] := [X, \sigma, \pi]$
4: **for** $k = 1$ **to** ILS-MAX **do**
5: Iter := 1
6: **while** Iter $\leq$ LS-MAX **do**
7: Compute a neighboring giant sequence: $\sigma' := Insertion[\sigma]$
8: Build a solution from σ' thanks to split: $X' := Split[\sigma', \pi]$
9: **if** Cycle time of $X' \leq$ Cycle time of X **then**
10: $[X, \sigma] := [X', \sigma']$
11: **if** Cycle time of $X' <$ Cycle time of X **then**
12: Iter := 0
13: $\pi :=$ LocalSearchOnRobots$[X, \sigma, \pi,$ Max-Neighbors-1$]$
14: **else**
15: $\pi :=$ LocalSearchOnRobots$[X, \sigma, \pi,$ Max-Neighbors-2$]$
16: **end if**
17: **end if**
18: Iter := Iter + 1
19: **end while**
20: **if** Cycle time of $X \leq$ Cycle time of X^* **then**
21: $[X^*, \sigma^*, \pi^*] := [X, \sigma, \pi]$
22: **end if**
23: Apply perturbation: $\sigma := Perturbation[\sigma^*]$
24: Update the solution: $X := Split[\sigma, \pi]$
25: **end for**
26: **return** X^*

(a) Keep the robot r on the workstation that has the biggest workload among the workstations that use r.
(b) For all the other workstations using r, choose the best robot not already assigned.

(3) Extract the permutation of robots from X to use it as an initial permutation of robots π.

Algorithm 3 relies on a local search on the permutations of robots with a fixed giant sequence of operations (lines 13, 15). It is described in Algorithm 4. In the latter, a swap move is used to generate a neighboring permutation of robots. It consists of selecting randomly two robots in the permutation and swapping them.

The split-based resolution method is experimented for both variants of the problem in the next section.

Algorithm 4 LocalSearchOnRobots

INPUT X, σ, π, Max-Neighbors: respectively a solution, its corresponding giant sequence of operations, its corresponding permutation of robots and a maximum number of visited neighbors.
OUTPUT π: A local-optimal permutation of robots.

```
Iter := 0
while Iter ≤ Max-Neighbors do
   Compute a neighboring permutation of robots: π' := Swap[π]
   Build a solution from (σ, π') thanks to split: X' := Split[σ, π']
   if Cycle time of X' ≤ Cycle time of X then
      [X, π] := [X', π']
      if Cycle time of X' < Cycle time of X then
         Iter := Max-Neighbors
      end if
   end if
   Iter := Iter + 1
end while
return π
```

6. Computational experiments

In this section, we conduct experiments in order to evaluate the split-based resolution method on benchmark instances from literature. The experiments were held on a basic computer equipped with a 8GB in RAM and a 1.6 GHz Intel Core i5 processor. Algorithms were implemented with JAVA 8 using ECLIPSE.

We consider the well-known instances of the Robotic Assembly Line Balancing Problem from [Gao et al.(2009)]. Those instances do not include sequence-dependent setup times. Setups were lately added by [Janardhanan et al.(2019)]. The authors suggest two new sets of instances derived from [Gao et al.(2009)]: instances with low setup times and instances with high setup times. Low setup times are generated (randomly and uniformly) between 0 and $0.25 \times min_{i,r}\{d_i^r\}$ while high setup times are generated between 0 and $0.75 \times min_{i,r}\{d_i^r\}$.

To the best of our knowledge, [Janardhanan et al.(2019)] is the only study published in literature dealing with the robotic assembly line balancing problem with sequence-dependent setup times with the objective of minimizing the cycle time. This latter study considers the second variant of the problem. It is used to evaluate our method in the next subsection.

6.1. *Second variant of the problem*

Several population-based metaheuristics are suggested in [Janardhanan et al.(2019)]. The authors use the same encoding technique than ours (giant sequence + permutation of robots) but use a different decoding procedure. This latter is greedy and described as A heuristic in [Janardhanan et al.(2019)]. It is of exponential-time complexity. Thus, it is interesting to compare our results with [Janardhanan et al.(2019)] to decide the relevance of using split as a decoding procedure.

[Janardhanan et al.(2019)] perform 30 random replications for each suggested metaheuristic. The best obtained results are given in their paper. The authors explicitly give

the value of the obtained cycle time only for instances with the number of operations up to 70. Only those instances are considered in this subsection. For bigger instances, [Janardhanan et al.(2019)] give another indicator from which we cannot derive the cycle time.

The experiments in [Janardhanan et al.(2019)] are performed on a super computer composed of a cluster of computers with a total capacity exceeding 60GB in RAM while we use a basic computer with 8GB in RAM. Therefore, it is not relevant to compare the CPU times of our method with those of [Janardhanan et al.(2019)] due this big gap in computing capacity. Our method converges in few minutes for most instances. The detailed CPU times are given in Table 5. They are roughly the same for instances without setup times, with low setup times or with high setup times.

Table 5. Approximate CPU times.

Instance		CPU time
n	$s_{max}(= n_r)$	in seconds
11	4	0
25	3	1
	4	1
	6	3
	9	4
35	4	7
	5	13
	7	22
	12	40
53	5	82
	7	128
	10	190
	14	245
70	7	600
	10	1100
	14	1900
	19	2300

The split-based metaheuristic depends on the parameters that are used in Algorithm 3. These parameters were fixed experimentally. They are given in Table 6.

Table 6. Split-based metaheuristic parameters.

ILS-MAX	10
LS-MAX	$n^2 * 5$
Max-Neighbors-1	n_r^2
Max-Neighbors-2	n_r

Table 7 shows the results on instances without setup times. The objective values from [Janardhanan et al.(2019)] are given in the table. They stand for the minimum cycle time over 30 random replications of the method. We also give the minimum cycle time over 30 random replications of our split-based iterated local search. The maximum, the mean and the standard deviation are also given. Comparison with [Janardhanan et al.(2019)] should be done by considering the min column. For each instance, if a method gives a better solution, it is highlighted in bold. Results on instances with low and high setup times are given in Tables 8 and 9.

We notice from Tables 7, 8 and 9 that, generally, our method gives better results than [Janardhanan et al.(2019)]. We also notice that the relative performance of the method improves as the instances get bigger. Besides, the split-based method performs better on instances with setup times relatively to [Janardhanan et al.(2019)]. It is even more effective as the setup times get bigger. We could explain the latter by the fact

Table 7. Results on instances without setup times.

Instance		[Janardhanan et al.(2019)]	Our results			
n	$s_{max}(=n_r)$	min	min	max	mean	Std deviation
11	4	**128**	**128**	128	128.0	0
25	3	**503**	**503**	520	505.26	5.77
	4	**327**	**327**	331	329.46	1.25
	6	**213**	**213**	214	213.16	0.37
	9	**121**	123	127	124.93	1.15
35	4	**449**	450	461	453.23	2.02
	5	**344**	**344**	377	358.73	14.08
	7	**222**	**222**	238	232.83	4.75
	12	**112**	**112**	116	113.8	1.37
53	5	559	**554**	562	556.16	2.03
	7	**320**	**320**	327	323.5	2.86
	10	239	**230**	247	239.46	5.11
	14	162	**158**	165	163.26	1.82
70	7	**448**	451	469	456.93	4.95
	10	**271**	**271**	281.0	275.6	2.61
	14	201	**199**	207	203.43	1.89
	19	152	**151**	157	153.5	1.5

Table 8. Results on instances with low setup times.

Instance		[Janardhanan et al.(2019)]	Our results			
n	$s_{max}(=n_r)$	min	min	max	mean	Std deviation
11	4	**137**	**137**	138	137.03	0.17
25	3	**516**	**516**	535	517.03	3.38
	4	**346**	**346**	347	346.03	0.17
	6	**227**	**227**	227	227	0
	9	131	**130**	132	131.7	0.64
35	4	462	**458**	487	465.16	7.00
	5	**355**	**355**	392	375.36	13.31
	7	237	**233**	246	241.63	3.80
	12	**118**	**118**	123	120.03	1.04
53	5	574	**570**	580	571.63	2.56
	7	334	**331**	337	334.03	1.88
	10	256	**242**	259	251.06	5.97
	14	170	**168**	175	172.6	1.40
70	7	469	**464**	484	472.26	4.95
	10	282	**280**	295	287.23	3.59
	14	211	**206**	216	212.46	2.70
	19	158	**156**	165	160.4	1.66

Table 9. Results on instances with high setup times.

Instance		[Janardhanan et al.(2019)]	Our results			
n	$s_{max}(=n_r)$	min	min	max	mean	Std deviation
11	4	**152**	**152**	171	160.1	5.71
25	3	**579**	**579**	590	582.13	3.82
	4	**380**	**380**	399	382.4	5.31
	6	**242**	**242**	248	245.43	2.80
	9	142	**141**	146	143.36	1.22
35	4	494	**487**	512	498.36	7.48
	5	392	**388**	430	403.7	12.62
	7	261	**259**	276	265.03	4.85
	12	131	**129**	136	132.63	1.81
53	5	619	**614**	633	622.96	6.24
	7	359	**357**	372	364.26	5.01
	10	**276**	277	285	279.83	2.17
	14	**185**	**185**	194	191.43	2.13
70	7	507	**492**	519	508.4	6.71
	10	309	**302**	318	309.8	4.10
	14	233	**226**	240	232.93	3.01
	19	175	**173**	181	176.7	1.67

that the heuristic decoder used in [Janardhanan et al.(2019)] is a greedy heuristic. It

assigns the operations greedily to workstations which does not give the optimal solution respecting a giant sequence when sequence-dependent setup times are considered.

We also notice that the standard deviation of the split-based method is low which demonstrates the robustness of the method.

We notice from Table 5 that the CPU times of the suggested method are low for instances with low number of operations and low number of robots. However, we notice that CPU times increase drastically for higher number of operations or higher number of robots. This is due to the complexity of the split algorithm that depends on these two factors. This can be cited as a limit of the method.

6.2. *First variant of the problem*

No author has already studied the first variant of the problem with sequence-dependent setup times. Only papers dealing with the basic Robotic Assembly Line Balancing Problem (without setup times) are available. Since our method could also be applied to null setup times, we choose to compare our method with latest resolution methods from the literature of the basic RALBP. We must note however that our method is not dedicated to this particular problem. Two latest resolution approaches were selected from literature, one exact method and one approximate method:

- [Borba, Ritt, and Miralles(2018)]: it is an exact approach extending the branch and bound from [Sewell and Jacobson(2012)] to the RALBP without setup times. This method has been shown to be efficient and solves to optimality instances with a number of operations going up to 89.
- [Nilakantan et al.(2015)]: it is an approximate method implementing a bio-inspired metaheuristic. The same encoding technique (giant sequence) is used. The authors use A heuristic to decode the giant sequence. A particular attention should be carried out to comparison with this method since it can evaluate the relevance of using split as a decoding procedure.

We recall that for this particular assumption, only a giant sequence is used to encode a solution since split is able to compute an optimal permutation of robots and an optimal balancing given a fixed giant sequence. 10 iterated local searches are performed. For each local search, the maximum number of visited neighbors is $n^2 * 5$. Only one random replication of the method is considered.

Table 10 gives the results on instances without setup times. For each method, the cycle time is given in the table.

Table 10. No setup instances.

Instance		[Nilakantan et al.(2015)]	[Borba, Ritt, and Miralles(2018)] (optimal solution)	Our results
n	$s_{max}(= n_r)$			
25	3	**503**	**503**	**503**
	4	327	**291**	**291**
	6	200	**194**	**194**
	9	110	**109**	**109**
35	4	**341**	**341**	**341**
	5	332	**329**	**329**
	7	211	**201**	**201**
	12	103	**93**	**93**
53	5	**449**	**449**	**449**
	7	294	**283**	**283**
	10	221	**203**	**203**
	14	142	**134**	**134**

We notice from Table 10 that, even if the method presented in this paper is not dedicated to the RALBP without setup times, it retrieved all the considered optimal solutions from [Borba, Ritt, and Miralles(2018)]. Besides, the split-based method clearly outperforms the population-based metaheuristic from [Nilakantan et al.(2015)].

Table 11 gives the results on instances with low and high setup times.

Table 11. Results on instances with low and high setup.

Instance		Low setup	High setup
n	$s_{max}(=n_r)$		
25	3	516	579
	4	310	343
	6	206	215
	9	117	121
35	4	352	376
	5	335	365
	7	208	222
	12	100	113
53	5	461	486
	7	289	314
	10	213	241
	14	143	155

As expected, we notice from Table 11 that the objective values (cycle times) get higher as setup times grow. The latter justifies that sequence-dependent setup times cannot be neglected and should be considered in the model of the Robotic Assembly Line Balancing Problem.

The CPU times of the suggested approach for the first variant are given in Table 12.

Table 12. Approximate CPU times.

Instance		CPU time
n	$s_{max}(=n_r)$	in seconds
11	4	<0.1
25	3	0.1
	4	0.3
	6	1.0
	9	1.5
35	4	4.1
	5	6.5
	7	7
	12	13
53	5	32
	7	44
	10	65
	14	92

7. Conclusion and perspectives

We have studied in this paper two variants of the robotic assembly line balancing problem with sequence-dependent setup times. A new approach is presented. This hybrid approach relies on an optimal path algorithm (split) intended to solve the particular case of a given giant sequence of operations. Split is then integrated in a metaheuristic framework. This encoding-decoding technique significantly reduces the size of the space search and preserves an optimal solution.

An experimental study is conducted on literature data for both variants of the problem. The considered instances have different characteristics (without setup times, with low setup times and with high setup times). The results show that the split-based approach clearly outperforms at least two methods from literature: [Nilakantan et al.(2015)] and [Janardhanan et al.(2019)].

From a managerial insight, this research offers to the decision-maker a decision support tool that can help addressing jointly the balancing, the robot selection, and the operation sequencing problems. The suggested method is simple and generic. New industrial constraints can be integrated easily.

This research offers many perspective research directions that could be categorized within two classes:

- Perspectives relative to the split-based approach:
  - Split can be embedded in more sophisticated metaheuristics. In a genetic algorithm for example, a chromosome can be encoded as a giant sequence and split can be used as a fitness evaluation function.
  - It can be quite beneficial for future research to accelerate split. This can be done by setting up new upper bounds that can be used to prune unpromising labels.
- Perspectives relative to application problems:
  - We focus in this paper on an efficiency-driven objective. However, it can be easy to study another objective together with the cycle time, for example the number of stations. Indeed, the labels used in split represent non-dominated solutions relative to these two objectives.
  - Robots are quite beneficial in the context of assembly lines but their energetic consumption is a real industrial issue. Split could be investigated for the objective of minimizing the energetic consumption.
  - The cooperation of humans and robots in the context of assembly lines is a hot topic in industry and research. It raises new objectives like ergonomics that can be investigated by means of a split-based approach.

Data Availability Statement

The data that support the findings of this study are available from the corresponding author, Y.L., upon reasonable request.

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

[Abdous(2019)] Abdous, Mohammed Amine. 2019. "Optimal design of manufacturing systems with ergonomics: application to assembly lines." PhD diss., Lyon.

[Allahverdi et al.(2008)] Allahverdi, Ali, CT Ng, TC Edwin Cheng, and Mikhail Y Kovalyov. 2008. "A survey of scheduling problems with setup times or costs." *European journal of operational research* 187 (3): 985–1032.

[Andres, Miralles, and Pastor(2008)] Andres, Carlos, Cristobal 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 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.

[Borba and Ritt(2014)] Borba, Leonardo, and Marcus Ritt. 2014. "A heuristic and a branch-and-bound algorithm for the assembly line worker assignment and balancing problem." *Computers & Operations Research* 45 (1): 87–96.

[Borba, Ritt, and Miralles(2018)] Borba, Leonardo, Marcus Ritt, and Cristóbal Miralles. 2018. "Exact and heuristic methods for solving the robotic assembly line balancing problem." *European Journal of Operational Research* 270 (1): 146–156.

[Chechik et al.(2016)] Chechik, Shiri, Haim Kaplan, Mikkel Thorup, Or Zamir, and Uri Zwick. 2016. "Bottleneck paths and trees and deterministic graphical games." In *33rd Symposium on Theoretical Aspects of Computer Science (STACS 2016)*, Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.

[Çil, Mete, and Ağpak(2016)] Çil, Zeynel Abidin, Süleyman Mete, and Kürşad Ağpak. 2016. "A goal programming approach for robotic assembly line balancing problem." *IFAC-PapersOnLine* 49 (12): 938–942.

[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.

[Gao et al.(2009)] Gao, Jie, Linyan Sun, Lihua Wang, and Mitsuo Gen. 2009. "An efficient approach for type II robotic assembly line balancing problems." *Computers & Industrial Engineering* 56 (3): 1065–1080.

[Janardhanan et al.(2019)] Janardhanan, Mukund Nilakantan, Zixiang Li, Grzegorz Bocewicz, Zbigniew Banaszak, and Peter Nielsen. 2019. "Metaheuristic algorithms for balancing robotic assembly lines with sequence-dependent robot setup times." *Applied Mathematical Modelling* 65: 256–270.

[Lahrichi et al.(2020)] Lahrichi, Youssef, Laurent Deroussi, Nathalie Grangeon, and Sylvie Norre. 2020. "A min-max path approach for balancing robotic assembly lines with sequence-dependent setup times." In *International Conference on Modeling, Optimization and Simulation (MOSIM)*, 10.

[Lahrichi et al.(2021)] Lahrichi, Youssef, Nathalie Grangeon, Laurent Deroussi, and Sylvie Norre. 2021. "A new split-based hybrid metaheuristic for the reconfigurable transfer line balancing problem." *International Journal of Production Research* 59 (4): 1127–1144.

[Levitin, Rubinovitz, and Shnits(2006)] Levitin, Gregory, Jacob Rubinovitz, and Boris Shnits. 2006. "A genetic algorithm for robotic assembly line balancing." *European Journal of Operational Research* 168 (3): 811–825.

[Miralles et al.(2008)] Miralles, Cristóbal, José P García-Sabater, Carlos Andrés, and Manuel Cardós. 2008. "Branch and bound procedures for solving the assembly line worker assignment and balancing problem: Application to sheltered work centres for disabled." *Discrete Applied Mathematics* 156 (3): 352–367.

[Nilakantan et al.(2015)] Nilakantan, J Mukund, Sivalinga Govinda Ponnambalam, N Jawahar, and Ganesan Kanagaraj. 2015. "Bio-inspired search algorithms to solve robotic assembly line balancing problems." *Neural Computing and Applications* 26 (6): 1379–1393.

[Prins(2004)] Prins, Christian. 2004. "A simple and effective evolutionary algorithm for the vehicle routing problem." *Computers & operations research* 31 (12): 1985–2002.

[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.

[Rubinovitz, Bukchin, and Lenz(1993)] Rubinovitz, Jacob, Joseph Bukchin, and Ehud Lenz. 1993. "RALB–A heuristic algorithm for design and balancing of robotic assembly lines." *CIRP annals* 42 (1): 497–500.

[Salveson(1955)] Salveson, Melvin E. 1955. "The assembly line balancing problem." *The Journal of Industrial Engineering* 18–25.

[Scholl, Boysen, and Fliedner(2013)] Scholl, Armin, Nils Boysen, and Malte Fliedner. 2013. "The assembly line balancing and scheduling problem with sequence-dependent setup times: problem extension, model formulation and efficient heuristics." *OR spectrum* 35 (1): 291–320.

[Sewell and Jacobson(2012)] Sewell, Edward C, and Sheldon H Jacobson. 2012. "A branch, bound, and remember algorithm for the simple assembly line balancing problem." *INFORMS Journal on Computing* 24 (3): 433–442.

[Yoosefelahi et al.(2012)] Yoosefelahi, A, M Aminnayeri, H Mosadegh, and H Davari Ardakani. 2012. "Type II robotic assembly line balancing problem: An evolution strategies algorithm for a multi-objective model." *Journal of Manufacturing Systems* 31 (2): 139–151.