A Polynomial Algorithm For Balancing a Sequence of Operations in Reconfigurable Transfer Lines

Youssef Lahrichi, Laurent Deroussi, Nathalie Grangeon, Sylvie Norre

A Polynomial Algorithm For Balancing a Sequence of Operations in Reconfigurable Transfer Lines

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

LIMOS CNRS UMR 6158, Aubière, France (e-mail: {youssef.lahrichi,laurent.deroussi,nathalie.grangeon,sylvie.norre}@uca.fr)

Abstract: We consider the problem of balancing reconfigurable transfer lines. The problem is quite recent and motivated by the growing need of reconfigurability in the industry. The problem consists into allocating a set of operations (necessary to machine a single part) to different workstations placed into a serial line. The workstations can contain multiple machines operating in parallel. The machines considered are mono-spindle head CNC machines which imply 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 suggest a polynomial exact algorithm that balances the transfer line provided the overall sequence of the operations (called "giant sequence") is given. This balancing subproblem was dealt with in the literature by means of ILP and heuristics. We use this algorithm to solve the balancing problem independently from the overall sequence of operations by embedding it in a simple local search within the giant sequence space. Experimentation show significant improvement compared to literature.

Keywords: Transfer line balancing, Dynamic programming, Split algorithm, Local search.

1. INTRODUCTION

New consuming trends, global competition and growing variety in demand in the actual economical context raises 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 workstations 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 refereed to as scalability Koren et al. (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, setup times between operations must be considered in addition to processing times in order to perform tool changing.

In section 2 we define the problem and introduce basic notations. The related work is described in section 3. We introduce a polynomial algorithm to balance a sequence of operations (subproblem) in section 4 and incorporate it in a simple local search to solve the general problem in section 5. An experimental study was also conducted. It is covered in section 6 of the paper.

2. RECONFIGURABLE TRANSFER LINE BALANCING PROBLEM

The Reconfigurable Transfer Line Balancing problem, denoted: RTLB, is a combinatorial optimisation problem whose instance could be described by the following data:

- The 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 constraints.

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:

⋆ 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).

- 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: they link two operations that must be assigned to the same workstation.
- Exclusion constraints must be respected: they link operations that could not be assigned to the same workstation.
- Accessibility constraints must be respected: every 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 a common position. For example, suppose we have 4 operations denoted o_1, o_2, o_3, o_4 and 6 possible part positions $A = \{0, 1, 2, 3, 4, 5\}$ such that

$$Pos(o_1) = \{0, 1, 2\}$$
$$Pos(o_2) = \{1, 2, 3\}$$
$$Pos(o_3) = \{2, 3, 4\}$$
$$Pos(o_4) = \{3, 4, 5\}$$

  Operations $\{o_1, o_2, o_3\}$ could be assigned to the same workstation because the position 2 is common between o_1, o_2 and o_3. However operations $\{o_1, o_2, o_4\}$ could not be assigned to the same workstation because there is no common position between o_1, o_2 and o_4.

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

Table 1. Table of notations

N	Set of operations, indexed on $\{1, 2, \ldots, n\}$
S	Set of workstations, indexed on $\{1, 2, \ldots, s_{max}\}$, s_{max} denotes the maximum number of workstations
P	Set of couples $(i, j) \in N \times N$ such that i precedes j (also denoted $i << j$)
M_{max}	Maximum number of machines in a workstation
N_{max}	Maximum number of operations in a workstation
C	Cycle time.
d_i	Processing time of operation i.
$t_{i,j}$	Set-up time to be considered when operation i is performed just before operation j in a workstation
I	Set of couples $(i, j) \in N \times N$ linked with an inclusion constraint
E	Set of couples $(i, j) \in N \times N$ linked with an exclusion constraint
Pos	Set of all possible part positions.
$Pos(i)$	Subset of part positions under what the operation i could be processed.

3. RELATED WORK

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:

- The consideration of 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.
- The consideration of 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 et al. (2008). The problem is studied in Martino and Pastor (2010).
- The consideration of inclusion, exclusion and accessibility constraints. The constraints come from the machining industry. Balancing problems considering those constraints are known as transfer line balancing problems. Many authors deal with this problem as shown in Battaïa and Dolgui (2013).

The RTLB problem consists of simultaneously solving two problems:

- The balancing problem: Assign the operations to the workstations.
- The sequencing problem: sequence the operations assigned to each workstation.

Therefore, resolution approaches for the RTLB 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. (2010c) suggest a MIP approach while Borisovsky et al. (2012) use a set partitioning model. We also cite Essafi et al. (2010b) and Essafi et al. (2010a) as integrated approaches. We have suggested an ILP in Lahrichi et al. (2018).
- Balance-First Sequence-Last (BFSL) methods: the sequencing step is done after the balancing step. We suggested an approximation algorithm of this type in Lahrichi et al. (2018). To the best of our knowledge no other approaches of type BFSL have been suggested.
- Sequence-First Balance-Last (SFBL) methods: the balancing step is done after the sequencing step. The problem of balancing a sequence of operations in a reconfigurable transfer line refers to the problem of

solving the RTLB given an overall sequence of all operations (giant sequence). It is a balancing subproblem consisting 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. It was dealt with in the literature by means of ILP (Borisovsky et al. (2012), Delorme et al. (2016)) and heuristics (Borisovsky et al. (2012)). In Borisovsky et al. (2012), a chromosome is coded as a giant sequence of all operations, either a heuristic decoder or a MIP is suggested to build up a solution. In Delorme et al. (2016), a multi-objective algorithm is suggested 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 split it using the MIP from Borisovsky et al. (2012) minimising the line cost.

To the best of our knowledge, a polynomial algorithm optimally splitting a giant sequence is not yet known which justify our contribution: a shortest path algorithm (called split) for solving this subproblem (section 4). We tested the efficiency of the split by incorporating it in a simple local search to solve the RTLB problem (Section 5).

4. A POLYNOMIAL ALGORITHM TO SOLVE THE BALANCING SUBPROBLEM

We first model the balancing subproblem as a constrained shortest path and construct the underlying graph. Then, we design a polynomial algorithm to compute this shortest path.

4.1 Construction of the graph

Given a *giant sequence* of operations $1, \ldots, n$, we consider the subproblem of balancing the line respecting the giant sequence.

For example, if the giant sequence is: $1, 2, 3, 4, 5, 6$. Two balancing solutions respecting the giant sequence could be

- Assigning the subsequence $1, 2, 3$ to the first workstation and $4, 5, 6$ to the second workstation.
- Assigning the subsequence $1, 2$ to the first workstation, $3, 4$ to the second workstation and $5, 6$ to the third workstation.

We claim that solving the balancing subproblem minimising the number of machines is equivalent to find the shortest weighted path of length inferior than (or equal to) s_{max} arcs between fictitious vertex 0 and the vertex n in the directed weighted graph

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

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 $i + 1, \ldots, j$. 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).

Some arcs (i, j) violating the maximum number of machines per workstation, the maximum number of operations per workstations, the inclusion, exclusion and accessibility constraints are deleted according to the following rules.

The maximum number of operations per workstation: N_{max}

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

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

The maximum number of machines per workstation: M_{max}

Likewise, an arc (i, j) violating the maximum number of machines per workstation constraint can be detected by the following condition:

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

Inclusion constraints

An arc (i, j) violating the inclusion constraints can be detected by the following condition:

$$\exists (a, b) \in I, (a, b) \notin \{i + 1, ..., j\}^2$$

Exclusion constraints

An arc (i, j) violating the exclusion constraints can be detected by the following condition:

$$\exists (a, b) \in E, (a, b) \in \{i + 1, ..., j\}^2$$

Accessibility constraints

An arc (i, j) violating the accessibility constraints can be detected by the following condition:

$$\exists (a, b) \in N \times N, (a, b) \in \{i+1, ..., j\}^2, Pos(a) \cap Pos(b) = \emptyset$$

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

We suggest the split algorithm (algorithm 1) in order to compute the shortest path. It uses labels on nodes to encompass information on the system state. We define a set of labels L_i for node i. The dominance rule that we describe afterwards limits the number of labels per node to s_{max}. Every label represents a path (partial solution) between 0 and i. A label 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.

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 $(a, b) \in$

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:

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

The shortest path cost is stored in *Shortest_Path_Cost*.

Algorithm 1 *split*

```
1: L_0 := {(0,0)}
2: for t=1 to n do
3:    L_t := ∅
4: end for
5: for t=0 to n-1 do
6:    for all (t,i) ∈ A (Propagate labels from L_t) do
7:       for all (a_t,b_t) ∈ L_t do
8:          if (b_t < s_max − 1 or i = n) then
9:             if (a_t + c_{t,i}, b_t + 1) is not dominated by an element of L_i then
10:               L_i := L_i ∪ {(a_t + c_{t,i}, b_t + 1)}
11:               if (a_t+c_{t,i}, b_t+1) dominates some element (a_i,b_i) ∈ L_i then
12:                  L_i := L_i\{(a_i,b_i)}
13:               end if
14:            end if
15:         end if
16:      end for
17:   end for
18: end for
19: if L_n = ∅ then
20:   Shortest_Path_Cost := +∞
21: else
22:   Shortest_Path_Cost := Min_{(a_i,b_i)∈L_n}(a_i)
23: end if
```

We describe the dominance rule as follows:

Definition 1. (Dominance rule) (a, b) is dominated by (a', b') if:

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

Lemma 2. 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. 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 3. 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 $m = n + (n - 1) + ... + 1 = \frac{n(n+1)}{2}$, split runs in $O(n^4)$.

5. A SIMPLE LOCAL SEARCH EMBEDDING THE SPLIT ALGORITHM

We incorporate the split into a simple local search to test its efficiency. The procedure uses the notion of compatible giant sequence. This notion is described in the next subsection then the local search will be detailed in the following subsections.

5.1 Computing a compatible giant sequence

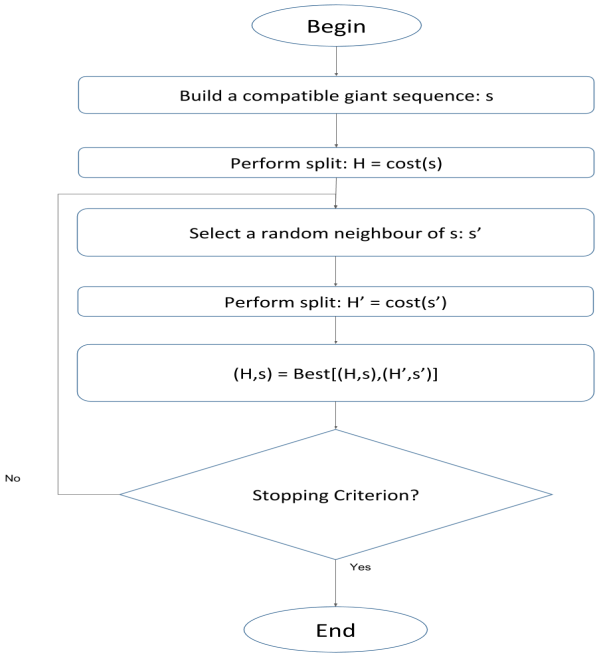

Fig. 1. *General scheme of the simple local search embedding the split algorithm*

A compatible giant sequence is a giant sequence for which there exist a feasible balancing solution respecting the giant sequence.

It is possible to test if a giant sequence is compatible with respect to an instance by applying split, it returns a positive (and optimal) shortest path cost only and only if the giant sequence is compatible.

If no inclusion, exclusion, accessibility or the maximum number of workstations constraints are taken into consideration, it is easy to compute a compatible giant sequence. Indeed, any giant sequence respecting the precedence constraints is compatible.

If we consider either inclusion, exclusion, accessibility or the maximum number of workstations constraints, any sequence respecting the precedence constraints is no longer guarantied to be compatible.

We define a weakly compatible giant sequence as a giant sequence for which there exists a balancing solution that is feasible with respect to all the constraints except the maximum number of workstations, that can be exceeded.

It is quite challenging to build up a compatible giant sequence; however a weakly compatible giant sequence could be built thanks to the following algorithm:

Algorithm to compute a weakly compatible giant sequence

(1) Gather the operations according to inclusion constraints. This step partitions the operations set into subsets:
$$S_1, S_2, \ldots, S_k$$
(2) Optimally sequence the operations in the subsets S_i (ATSP problem, Held and Karp (1962)).
(3) A weakly compatible giant sequence σ is then computed:
   (a) Select randomly a set S_i such that all predecessors of operations in S_i are already contained in σ.
   (b) Append S_i to σ.
   (c) Return to (a) while σ is incomplete.

The giant sequence given by the previous algorithm is weakly compatible. Indeed, a feasible solution (eventually violating the max. number of workstations constraint) could be obtained by applying a modified version of the split allowing solutions exceeding s_{max} workstations (delete line 8 in algorithm 1).

After obtaining a weakly feasible solution, a repair procedure is designed to obtain a feasible solution. It consists in merging the workstations in order to reduce the number of workstations and therefore having a feasible solution.

The repair procedure works as follows:

- Choosing randomly two workstations WS_1 and WS_2.
- Merge WS_1 and WS_2 into one workstation WS if it does not violate any constraint.
- Operations in WS are resequenced optimally (Asymmetric travelling salesman problem) thanks to dynamic programming (Held and Karp (1962)).

If no compatible giant sequence is found a different random seed is used. If the infeasibility persists, the compatible giant sequence is computed differently: going from a weakly compatible giant sequence, a local search with insertion neighbourhood is applied to get a compatible giant sequence. A neighbour is accepted if its split uses less workstations.

5.2 General scheme

The split algorithm could be used to solve the RTLB problem. Once having computed a starting giant sequence, we perform a local search in the space of the giant sequences S, the split is used at each iteration to evaluate this sequence.

The general scheme is depicted Fig. 1.

5.3 The neighbouring system

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 s:
$$s = (o_1, , o_n)$$
a random neighbour is selected by selecting a random operation $o_i, 1 \leq i \leq n$. Once this operation selected two operations must be identified $o_{i'}$ and $o_{i''}$ such that
$$i' = max\{j; j < i \ \& \ (j,i) \in P\}$$
and
$$i'' = min\{j; i < j \ \& \ (i,j) \in P\}$$
Then a random position is selected between i' and i'' to (re)insert operation i.

5.4 Cost evaluation and neighbour acceptance

The cost of a giant sequence is computed thanks to the split algorithm. Therefore, if s is a giant sequence, $cost(s)$ is the minimum number of machines necessary to respect s if it is compatible and $+\infty$ otherwise. To be accepted a neighbour must have a cost smaller or equal to the current cost (this defines $Best[(H,s),(H',s')]$ in Fig. 1).

6. EXPERIMENTAL RESULTS

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 (Borisovsky et al. (2013)). The authors mention on the paper the given objective value for each instance. The comparison is consequently 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: $7 \leq |I|, |E| \leq 15$
- Accessibility constraints are taken into consideration in these instances.

We can compute a starting compatible giant sequence for all the instances. Starting from this initial giant sequence, 10 independent runs of the local search are performed. We use the following notations in the table of experimentation:

- z_{lb}: the lower bound obtained with the linear relaxation of the ILP from Lahrichi et al. (2018).
- *cost*: cost (number of machines) of the initial solution obtained by applying split to the initial compatible giant sequence.
- WS: number of workstations of the initial solution
- GA: Best solution obtained by the genetic algorithm: Borisovsky et al. (2013). Results were obtained by taking the best result out of 10 independent runs for each instance. CPU time was bounded to 15 min for each run.
- min: minimum objective function obtained by our method over 10 independent runs.
- max: maximum objective function obtained by our method over 10 independent runs.
- mean: average objective function obtained by our method over 10 independent runs.
- σ: Standard deviation on objective function obtained by our method over 10 independent runs.

Table 2. Table of experimentation

Inst.	z_{lb}	GA	Initial sol.		our method (10 replications)			
			cost	WS	min	max	mean	σ
A1	23	33	40	25	**29**	**32**	30.7	0.78
A2	22	33	38	24	**28**	**31**	28.7	0.9
A3	22	31	36	24	**29**	**30**	29.1	0.3
A4	22	29	39	24	**28**	30	29	0.63
A5	-	32	40	25	**29**	**31**	30.1	0.83
A6	22	32	39	25	**29**	**32**	30.1	0.83
A7	23	34	38	23	**29**	**32**	30.5	1.2
A8*	-	31	-	-	-	-	-	-
A9	22	30	40	25	**29**	**30**	29.7	0.45
A10	24	32	41	25	**32**	33	32.5	0.5
A11	22	30	36	24	**28**	**30**	29.3	0.64
A12	23	31	39	24	**29**	**31**	30.4	0.66
A13	23	33	39	23	**30**	**31**	30.5	0.5
A14	22	31	38	24	**29**	**31**	30	0.77
A15	22	33	36	25	**29**	**30**	29.5	0.5

The CPU time was bounded was bounded with the same time than the GA (15 min), however the solutions were obtained in much fewer times. The comparison should be done between the GA column and the (our method) min column. The min column gives objective values 9.16% lower on average than GA. Besides, even the max values are lower than the GA for all the instance but two. We notice that the standard deviation is low which demonstrate the robustness of the method.

No solution has been found for A8 instance. It is shown to be infeasible. Inclusion and exclusion constraints are contradictory: two operations are linked at the same time with inclusion and exclusion constraints.

7. CONCLUSION

We show in this article that balancing a sequence of operations in reconfigurable transfer lines is a polynomial-time (P) problem by modelling it as a constrained shortest path problem and giving a polynomial algorithm (split) to solve it. We tested the efficiency of the split by incorporating it in a simple local search to solve the RTLB problem.

Experimentation on literature instances show very good results. The use of the split algorithm is relevant even with a basic metaheuristic. Several directions could be taken as a future research: (by order of importance)

- Incorporating the split in more sophisticated metaheuristic frameworks or a branch and bound.
- Study of the rebalancing problem when the product or the cycle time changes.
- Considering the bi-criteria optimisation by minimising the cycle time and the number of machines.
- Study the problem in the context of uncertainty.
- Considering mixed-model transfer line balancing problem.

REFERENCES

Andrés, C., Miralles, C., and Pastor, R. (2008). Balancing and scheduling tasks in assembly lines with sequence-dependent setup times. *European Journal of Operational Research*, 187(3), 1212 – 1223.

Battaïa, O. and Dolgui, A. (2013). A taxonomy of line balancing problems and their solution approaches. *International Journal of Production Economics*, 142(2), 259 – 277.

Borisovsky, P.A., Delorme, X., and Dolgui, A. (2012). Balancing reconfigurable machining lines by means of set partitioning model. *IFAC Proceedings Volumes*, 45(6), 426 – 431.

Borisovsky, P.A., Delorme, X., and Dolgui, A. (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, G.M. (1974). Assembly line balancing with multiple stations. *Manage. Sci.*, 20(6), 1010–1021.

Delorme, X., Malyutin, S., and Dolgui, A. (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.

Essafi, M., Delorme, X., and Dolgui, A. (2010a). Balancing lines with cnc machines: A multi-start ant based heuristic. *CIRP Journal of Manufacturing Science and Technology*, 2(3), 176–182.

Essafi, M., Delorme, X., and Dolgui, A. (2010b). Balancing machining lines: a two-phase heuristic. *Studies in Informatics and Control*, 19(3), 243–252.

Essafi, M., Delorme, X., Dolgui, A., and Guschinskaya, O. (2010c). A mip approach for balancing transfer line with complex industrial constraints. *Computers Industrial Engineering*, 58(3), 393 – 400.

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

Koren, Y. and Shpitalni, M. (2010). Design of reconfigurable manufacturing systems. *Journal of Manufacturing Systems*, 29(4), 130 – 141.

Koren, Y., Wang, W., and Gu, X. (2017). Value creation through design for scalability of reconfigurable manufacturing systems. *International Journal of Production Research*, 55(5), 1227–1242.

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

Martino, L. and Pastor, R. (2010). Heuristic procedures for solving the general assembly line balancing problem with setups. *International Journal of Production Research*, 48(6), 1787–1804.

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

Rabbani, M., Siadatian, R., Farrokhi-Asl, H., and Manavizadeh, N. (2016). Multi-objective optimization algorithms for mixed model assembly line balancing problem with parallel workstations. *Cogent Engineering*, 3(1), 115–203.

Vilarinho, P.M. and Simaria, A.S. (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.