Yuzuncu Yil University
Journal of the Institute of Natural & Applied Sciences

https://dergipark.org.tr/en/pub/yyufbed

Research Article

Benders Decomposition Algorithm for Solving the Type-II Simple Assembly Line Balancing Problem with Zoning Restrictions

Alper HAMZADAYI

Van Yuzuncu Yil University, Engineering Faculty, Department of Industrial Engineering, 65080, Van, Turkey

Alper HAMZADAYI, ORCID No: 0000-0003-4035-2775

Corresponding author e-mail: alperhamzadayi@yyu.edu.tr

Article Info

Received: 28.02.2022
Accepted: 14.06.2022
Online August 2022
DOI: 10.53433/yyufbed.1080238

Keywords
Assembly line balancing,
Benders decomposition,
Referenced local search,
Type-II,
Zoning restrictions

Abstract: This paper considers the type-II assembly line balancing problem under zoning constraints, where a number of tasks must be assigned to a number of workstations while respecting compatibility (or otherwise) between tasks and stations, and by observing the precedence relationships between tasks. The objective is to minimize the latest completion time at any station. The paper proposes an exact algorithm that takes into account an exact formulation of the problem as well as iteratively introduces a set of constraints in the spirit of Benders decomposition. In addition to boundary constraints on the decision variables, the algorithm makes use of combinatorial cuts and a referenced local search in order to generate upper bounds. The algorithm is extensively evaluated on benchmark instances, which indicates that it outperforms the state-of-the-art approaches to the problem.

Tip-II Basit Montaj Hattı Dengeleme Probleminin Bölge Kısıtlamaları ile Çözümüne Yönelik Benders Ayrıştırma Algoritması

Makale Bilgileri

Geliş: 28.02.2022
Kabul: 14.06.2022
Online Ağustos 2022
DOI: 10.53433/yyufbed.1080238

Anahtar Kelimeler
Benders ayrıştırması,
Bölge kısıtlamaları,
Montaj hattı dengeleme,
Referanslı yerel arama,
Tip-II

Öz: Bu makale, işler ve istasyonlar arasındaki uyumluluğu (veya başka türlü) dikkate alarak ve istasyonlar arasındaki öncelik ilişkilerini gözlemleyerek belirli bir iş istasyonu kümesine bir dizi iş atamaktan oluşan, bölgeleme kısıtlamaları ile tip II montaj hattı dengeleme problemi ile ilgilidir. Amaç, herhangi bir istasyondaki en son tamamlanma süresini en aza indirmektir. Bu makale, Benders ayrıştırmasının yapısına uygun olarak bir dizi kısıtlamayı yinelemeli olarak ortaya koyan bir problem formasyonuna dayanan kesin bir algoritmayı açıklamaktadır. Algoritma, üst sınırlar oluşturmak için karar değişkenleri, kombinatoryel kesimler ve referanslı bir yerel arama üzerinde bir dizi sınırlayıcı kısıtlama içerir. Algoritmanın problem için en gelişmiş yaklaşımlardan daha üstün olduğu kıyaslamalı örnekler üzerinde kapsamlı hesaplama deneyleri ile gösterilmiştir.

1. Introduction

Assembly lines are part of production systems and consist of successive workstations that are connected by a material handling system, and are used to perform a set of tasks on a product that flows through them. Each workstation must complete the tasks within a fixed duration called the cycle time, where each task has a particular execution time and precedence relationships with other tasks (Hamzadayı, 2018). The simple assembly line balancing problem (SALBP) is concerned with the

allocation of tasks amongst the workstations while respecting the required ordering of the tasks. The problem is said to be of type-I when the aim is to minimize the number of workstations in a predefined cycle time, and of type-II (hereafter denoted SALBP-II) when the goal is to minimize the cycle time by considering the given a predefined number of workstations. The SALBP is NP-Hard (Karp, 1972), following which numerous solution algorithms have been proposed (see, e.g., Scholl & Becker, 2006; Boysen et al., 2008). More relevant to the current paper are the mixed-integer linear programming (MILP) formulations and exact methods. To our knowledge, the first mathematical formulation as a 0-1 mathematical program is proposed by Bowman(1960), later modified by White (1961) by converting some integer variables into binary, and further improved by Patterson and Albracht (1975) a decade later where additional upper and lower bounds were described on the number of workstations to reduce the number of variables. An alternative general integer programming formulation appears in Talbot and Patterson (1984) solved using an adaptation of an algorithm by Balas (1965). It was shown earlier that for this problem the effectiveness of a formulation depends on the technique with which it is solved (Amen, 2006). A model with additional constraints on the indices of the workstations is presented by Pastor and Ferrer (2009) to dynamically reduce the search space as a function of the incumbent solution, which does not require an initial upper bound and uses the intermediary solutions to cut-off solutions from the feasible set. A more recent work by Ritt and Costa (2018) presents a stronger formulation for the problem including precedence constraints and limits on the number of workstations used, that improves upon Patterson and Albracht (1975) and Bowman (1960), yielding optimal solutions for instances not hitherto solved or identification of new best solution values. Other exact methods for the problem include iterative algorithms (Baybars, 1986; Scholl & Becker, 2006), dynamic programming (Held et al.,1963; Jackson, 1956; Schrage & Baker, 1978), and branch-and-bound (Scholl & Klein, 1997) that outperforms dynamic programming. The most prominent of the latter are FABLE (Johnson, 1988), EUREKA (Hoffmann, 1992), SALOME-1 (Scholl & Klein, 1997), task-oriented branch-and-bound (TBB) (Klein & Scholl, 1996) and SALOME-2 (Klein & Scholl, 1996). Of the existing methods, the formulation proposed by Ritt and Costa (2018) and SALOME-2 by Klein and Scholl (1996) seems to be the state-of-the-art in solving the SALBP. The existing exact methods for the SALBP-II are variants of those proposed for the type-I, namely FABLE, EUREKA, SALOME-1 and TBB, with SALOME-2 (Klein & Scholl, 1996) being the currently best-performing exact method that outperforms even the existing heuristic and metaheuristic algorithms for the same problem. One practical extension of the SALBP-II is the case where some tasks are compatible, meaning that they have to be assigned to the same workstation, or physically incompatible, indicating that they cannot be processed in the same workstation. Previous work on assembly line balancing problems with zoning constraints is limited (see, e.g., Klein, 1963; Gokcen & Erel, 1997; Vilarinho & Simaria, 2002; Baykasoglu & Dereli, 2008; Özcan & Toklu, 2009; Akpınar & Bayhan, 2011; Özbakır & Tapkan, 2011), wherein mathematical models and heuristic algorithms are proposed, but no exact techniques were described. The present paper describes, to the best of the authors' knowledge, the first exact algorithm for the SALBP-II with zoning restrictions. The algorithm employs the principles of Benders decomposition, a technique that has seen a limited number of applications to other variants of the assembly line balancing problems (Hazır & Dolgui, 2013 and 2015; Akpinar et al., 2017; Huang et al., 2021; Furugi, 2022). Operating on a formulation that appears in Baybars (1986), the algorithm introduces a new set of constraints on the domain of the decision variables at each iteration. We incorporate three different cut generation strategies and combinatorial cuts within the algorithm. Finally, we describe a referenced local search-based algorithm to obtain upper bounds for the problem. The paper reports the results of extensive computational experimentation undertaken to test the performance of the algorithm, which yields superior results on benchmark instances in comparison with the state-of-the-art on the problem.

The remainder of this paper is structured as follows. Section 2 formally defines the problem and presents two integer programming models. The proposed algorithm is described in detail in Section 3, and computational results are presented in Section 4, followed by conclusions in Section 5.

2. Integer Linear Programming Formulations

The SALBP-II can formally be described as the assignment of a set N of tasks, where each $i \in N$ has a set F_i of followers (or a set P_i of predecessors) and an execution time (or duration) of t_i units, to a set S of workstations in order to minimize the cycle time c defined as the latest completion time at any

station. In any feasible assignment to the problem, all tasks $j \in P_i$ should be assigned to a workstation that appears before the workstation to which task i is assigned. We also denote by set $N^+ = \{(i, j) \mid i \in N, j \in N \setminus \{i\}, i \parallel j\}$ the pair of tasks that need to be assigned to the same workstation, and by set $N^- = \{(i, j) \mid i \in N, j \in N \setminus \{i\}, i \nparallel j\}$ those pairs of tasks that cannot be assigned to the same workstation, where the operators $\parallel$ and $\nparallel$, denote compatibility and incompatibility, respectively, between distinct pairs of tasks. In what follows, we present extensions of two known formulations for the problem relevant to our work.

2.1. A natural formulation

The first formulation given below is an extension of that described by Baybars (1986), where an index set S of m workstations is given as input. The model uses a binary variable $x_{s,i}$ which is set equal to 1 if and only if task $i \in N$ is assigned to workstation $s \in S$, and to 0 otherwise.

Objective function:

$$\text{(BM) Minimize } c \tag{1}$$

Subject to:

$$\sum_{s \in S} x_{s,i} = 1 \;\; i \in N \tag{2}$$

$$x_{t,j} \leq \sum_{s \in S \mid s \leq t} x_{s,i} \quad i \in N, j \in F_i, t \in S \tag{3}$$

$$\sum_{i \in N} t_i x_{s,i} \leq c \quad s \in S \tag{4}$$

$$x_{s,i} - x_{s,j} = 0 \;\; (i,j) \in N+, s \in S \tag{5}$$

$$x_{s,i} + x_{s,j} \leq 1 \;\; (i,j) \in N-, s \in S \tag{6}$$

$$x_{s,i} \in \{0,1\} \; i \in N, s \in S \tag{7}$$

In this model, the objective function (1) minimizes the cycle time. Constraints (2) ensure that each task $i \in N$ is assigned to exactly one workstation $s \in S$. Constraints (3) ensure that tasks are assigned in accordance with the precedence relationships between them. More specifically, a task j in the set F_i of followers for task $i \in N$ assigned to a workstation $s \in S$ cannot be assigned to a workstation $t \geq s$. Constraints (4) collectively define the cycle time as the longest time spent at any workstation $s \in S$. Zoning restrictions are formulated by (5) and (6) for compatible and incompatible pairs of tasks, respectively. Finally, (7) impose integrity restrictions on the $x_{s,i}$ variables.

2.2. The formulation of Ritt and Costa (2018)

The following model denoted RC is an extension of the one proposed by Ritt & Costa (2018), which was computationally shown to be the best-performing formulation to date for the type-II SALBP. In particular, Ritt & Costa (2018) have tested this formulation on 302 standard benchmark instances of the problem and conclude that it achieves the largest number of best solutions in a short time, which is the reason why we adapt it to the problem at hand. The formulation uses the same variables as BM, and is given below.

Objective function:

$$\text{(BM) Minimize } c \tag{8}$$

Subject to:

$$\sum_{s \in S} x_{s,i} = 1 \;\; i \in N$$

$$\sum_{s \in S \mid s \leq t} x_{s,i} \;\geq \sum_{s \in S \mid s \leq t} x_{s,j} \qquad i \in N, j \in F_i, t \in S \tag{9}$$

$$\sum_{i \in N} t_i x_{s,i} \;\leq c \;\; s \in S$$

$$x_{s,i} - \; x_{s,j} = 0 \;\; (i,j) \in N+, s \in S$$

$$x_{s,i} + \; x_{s,j} \leq 1 \;\; (i,j) \in N-, s \in S$$

$$x_{s,i} \in \{0,1\} \; i \in N, E_i(U,m) \;\leq s \;\leq \; L_i(U,m) \tag{10}$$

The constraints of this formulation play the same role as in the previous one. One difference is in constraints (10), which limits the index of the workstations to which a task can be assigned. In particular, for a given cycle time c and a given number of workstations m, let $E_i(c,m)$ be the earliest and $L_i(c,m)$ the latest admissible workstation for task $i \in N$. For the SALBP, these bounds can be set to $E_i(c,m) = \left\lceil \sum_{j \mid j \leq i} t_j \,/c \right\rceil$ and $L_i(c,m) = m + 1 - \left\lceil \sum_{j \mid j \leq i} t_j \,/c \right\rceil$. Thus, task i can only be assigned to one of the workstations in the workstation interval $[E_i;L_i]$. In constraints (10), U is an upper bound for the cycle time, calculated as $U = 2L$ as a function of the lower bound $L = \max\{\mu(N), \lceil \delta(N)/m \rceil\}$ with $\mu(N) = \max_{i \in N}\{t_i\}$ and $\delta(N) = \sum_{i \in N} t_i$. Further details on this model are given by Ritt & Costa (2018).

3. The Proposed Exact Algorithm

The algorithm we propose in this paper uses a combination of Benders decomposition, additional cuts, and a referenced local search that generates upper bounds. All three ingredients are described in further detail below.

3.1. Application of Benders decomposition

The application of Benders decomposition (Benders, 1962) to the SALBP-II is as follows. Let $M(c, x)$ denote the formulation (1)–(7) where $x = \{x_{s,i} | i \in N, s \in S\}$. If the assignment variables are fixed as $x = \{\hat{x} | \hat{x} \text{ satisfies } (2), (3), (7)\}$, then the remaining problem $M(c, \hat{x})$ only entails finding the value of the variable c, and is always feasible. If $\alpha = \{\alpha_s \geq 0 | s \in S\}$ is the vector of dual variables corresponding to constraints (4), the dual (see constraints (11)) $D(\alpha, \hat{x})$ of $M(c, \hat{x})$ can be represented as which can be solved by inspection with an optimal solution α^* such that $\alpha^*_{s'} = 1$ where $s' = \operatorname{argmax}_{s \in S} \sum_{i \in N} t_i x_{s,i}$, and $\alpha^*_s = 0$ for all $s \neq s'$.

$$\text{Maximize} \sum_{s \in S} \alpha_s \sum_{i \in N} t_i \, \hat{x}_{s,i} \text{ subject to } \sum_{s \in S} \alpha_s \; = 1 \;\; \alpha \geq 0 \tag{11}$$

This gives rise to the following reformulation $M(P)$ of BM, called the master problem (see constraints (12) and (13)),

Objective function:

$$\text{Minimize } \vartheta \tag{12}$$

Subject to (2), (3), (5), (6), (7) and

$$\vartheta \geq \sum_{s \in S, i \in N} \alpha_s \, t_i \hat{x}_{s,i} \quad \alpha \in P \tag{13}$$

where P is the set of extreme points of $D(\alpha, \hat{x})$ and where (13) are referred to as the optimality cuts. Note that the master problem is no different from the original formulation (8) – (10), with the exception of constraint set (10) being replaced by the optimality cuts, and it may seem counter-intuitive to transform a polynomial-size formulation with an exponential size one. However, our algorithm solves the $M(P)$ iteratively, which implies that there will only be a single optimality cut added at each iteration, which is precisely the one that corresponds to station s'.

3.2. Additional inequalities

To help speed up the convergence of the algorithm, we add two different sets of cuts at each iteration of the algorithm, described as follows. The first set of inequalities restricts the decision variables given in constraints (7) for the workstation s' identified through the solution of the dual problem described above. This is done by first computing the earliest and the latest admissible workstation for each task as described in Section 2.2, and then adding the constraints (14) at each iteration for workstation s'.

$$x_{s',i} = 0 \; \forall i \in N \text{ such that } L_i(U,m) < s' \text{ or } s' < E_i(U,m) \tag{14}$$

The second set of inequalities are combinatorial cuts (Codato & Fischetti, 2006; Côté et al., 2014). In our implementation, we use combinatorial cuts to eliminate feasible solutions from the set of solutions to the problem. In particular, let $\Delta_{s'} \subseteq N$ be the set of all tasks that satisfy the condition $E_i(U,m) \leq s' \leq L_i(U,m)$ for workstation s'. Let $D_{s'} \subseteq \Delta_{s'}$ be a subset of task indices that can be assigned to workstation s' satisfying the precedence relationships, i.e., those that ensure a feasible assignment. We then employ a roulette wheel selection mechanism (Holland, 1975) in choosing a task $i^* \in D_{s'}$ by using one of the three strategies $\Gamma \in \{1,2,3\}$ below:

1. The probability of choosing a task is inversely proportional to its duration (Γ_1),
2. The probability of choosing a task is proportional to its duration (Γ_2),
3. Each task has a uniform selection probability (Γ_3).

Task i^* is added to the (initially empty) set $C_{s'}$, is removed from $\Delta_{s'}$ and set $D_{s'}$ is updated in line with the precedence relationships. We repeat the procedure until there is a task $i' \in N \backslash C_{s'}$ for which $\delta(C_{s'} \cup \{i'\}) > c$, i.e., the addition of task i' to set $C_{s'}$ results in a cycle time larger than c. In this case, task i' is added to set $C_{s'}$, and the following inequalities are the combinatorial cuts (see constraints (15)) we add to cut a solution off from the feasible set of solutions. We limit the number of cuts to be added by a user-defined parameter ς_C at each iteration.

$$\sum_{i \in C_{s'}} x_{s',i} \leq |C_{s'}| - 1 \tag{15}$$

Algorithm 1. Combinatorial cut generation for one workstation

1: **Input**: Workstation s', $C_{s'} \leftarrow \emptyset$, cut generating strategy Γ
2: $\Delta_{s'} = \{i \in N | L_i(U,m) < s' \text{ or } s' < E_i(U,m)\}$
3: **repeat**
4: Generate $D_{s'} \subseteq \Delta_{s'}$ as the set of tasks that meet the precedence constraints
5: Choose $i^* \in D_{s'}$ according to strategy Γ
6: **if** $\delta(C_{s'}) + t_{i^*} \leq c$ **then**
7: $C_{s'} \leftarrow C_{s'} \cup \{i^*\}$
8: $D_{s'} \leftarrow D_{s'} \backslash \{i^*\}$
9: **end if**
10: **until** $\delta(C_{s'}) + t_{i^*} > c$
11: **Output**: Set $C_{s'}$ to construct a combinatorial cut (15)

3.3. An upper bounding procedure

This section describes a referenced local search (RLS) to generate good-quality upper bounds for the problem and embedded within the exact algorithm. The reason behind the choice of the RLS, as opposed to alternative heuristics, is the simplicity of its implementation and reliance on only a single input parameter, which is generally equal to the number of tasks in the problem. The RLS has been successfully implemented to solve the various flow shop problems (Deng & Gu, 2012; Pan & Ruiz, 2014), and is, to our knowledge, being described for solving an assembly line balancing problem for the first time.

The RLS is a multi-start algorithm that first calculates a range $[\vartheta', \vartheta'']$ in which the cycle time c is to lie, where ϑ' is set equal to the theoretical lower bound of the problem and $\vartheta'' = \vartheta' + \rho$, where ρ is a step-size initially set equal to the average task execution time. The aim is first to find a cycle time c that ensures a feasible assignment of all tasks to the m workstations prescribed in the problem instance, calculated as in Algorithm 2.

Algorithm 2. Computing a feasible cycle time

1: **Input**: number m of workstations in the instance
2: Initialize: $\vartheta' = \max\{\mu(N), \lceil \delta(N)/m \rceil\}$, $\rho = \lceil \delta(N)/|N| \rceil$, $c = \vartheta''$
3: Set $\pi \leftarrow \pi^{ref}$, $\pi^* \leftarrow \pi$, $i = 1$, $m^* = 0$
4: **while** $m^* < m$ **do**
5: **for** each $j \in N$ **do**
6: Let π' be the sequence where task i is inserted into the position j in π
7: Calculate the number m' of workstations and the total idle time φ' for π'
8: **if** the solution satisfies the zoning constraints, **then**
9: **if** $m' \geq m$ **then**
10: $m^* = m$
11: **else**
12: **if** $m^* < m$ **then**
13: $\pi^* \leftarrow \pi'$, $m^* \leftarrow m'$, $\varphi^* \leftarrow \varphi'$
14: **else if** $m^* = m$ **then**
15: **if** $\varphi^* > \varphi'$ **then**
16: $\pi^* \leftarrow \pi'$, $\varphi^* \leftarrow \varphi'$
17: **end if**
18: **end if**
19: **end if**
20: **end if**
21: **end for**
22: **if** $m^* \geq m$ **then**
23: $c^* = c$
24: **end if**
25: $c = c + \rho$
26: $\pi \leftarrow \pi^*$
27: $i \leftarrow \mathrm{mod}(i + 1, |N|)$
28: **end while**
29: **Output**: A feasible cycle time c^*

If a given value of ϑ'' does not produce a feasible solution, then ϑ' is set equal to ϑ'', and ϑ'' itself is increased by the step-size ρ. Once a feasible cycle time is obtained, the algorithm then gradually

reduces the width of the interval $[\vartheta', \vartheta'']$ to try an obtain a better feasible cycle time that lies within this interval. This is done by setting the cycle time as $c = (\vartheta', \vartheta'')/2$. If this value results in a feasible solution, then c is set equal to ϑ'', otherwise it is set equal to ϑ'. Note that the ϑ'' always represents a cycle time value that provides a feasible solution. A pseudo-code of the RLS is given in Algorithm 3.

The RLS assumes an initial task sequence π^{ref} as input, which is generated using the precedence relationships in a given SALBP-II instance, such as the one shown in Figure 1. In the figure, each task is represented with a circle within which lies the task number, and above which is the execution time of that task. An arrow from task i to task j shows that task i has to be completed any time before the start time of task j. On this instance, we first generate a random sequence of tasks, say {3, 1, 4, 2, 7, 6, 5} that will be used to decide on the priority $p(i)$ of each task, namely $p(3) = 1$, $p(1) = 2$, $p(4) = 3$, $p(2) = 4$, and so on, where $p(i) > p(j)$ indicates that i should appear earlier in a feasible sequence. The construction of π^{ref} starts by considering the first position, for which task 1 is the only candidate. As for the second position, the precedence graph in Figure 1 indicates that tasks 2 and 4 are the only candidates, of which the former is chosen given that $p(2) > p(4)$. Continuing in a similar manner result in π^{ref} = {1, 2, 5, 6, 4, 7, 3}.

Algorithm 3. The referenced local search (RLS) algorithm

Input: number m of machines in the instance and a feasible cycle time c^*
Initialize: $\vartheta' = \max\{\mu(N), \lceil \delta(N)/m \rceil\}, \vartheta'' = c^*, \rho = \lceil (\vartheta'' - \vartheta')/2 \rceil, c = \vartheta' + \rho$
while $\rho > 1$ **do**
  Set $\pi \leftarrow \pi^{ref}$, $\pi^* \leftarrow \pi$, $i = 1$, $k = 1$, $m^* = 0$, $f_{sol} = 0$
  while $m^* < m$ or $k \leq N$ **do**
    for each $j \in N$ **do**
    Let π' be the sequence where task i is inserted into the position j in π
    Calculate the number m' of workstations and the total idle time φ' for π'
      if the solution satisfies the zoning constraints, **then**
        if $m' \geq m$ **then**
        $m^* = m$
        $f_{sol} = 1$
        else
          if $f_{sol} = 1$ **then**
            if $m^* > m$ **then**
            $\pi^* \leftarrow \pi', m^* \leftarrow m', \varphi^* \leftarrow \varphi'$
            else if $m^* = m$ **then**
              if $\varphi^* > \varphi'$ **then**
              $\pi^* \leftarrow \pi', \varphi^* \leftarrow \varphi'$
              end if
            end if
          else
            if $m^* < m$ **then**
            $\pi^* \leftarrow \pi', m^* \leftarrow m', \varphi^* \leftarrow \varphi'$
            else if $m^* = m$ **then**
              if $\varphi^* > \varphi'$ **then**
              $\pi^* \leftarrow \pi', \varphi^* \leftarrow \varphi'$
              end if
            end if
          end if
        end if
      end if
    end for
    $\pi \leftarrow \pi^*$
    $i \leftarrow \mathrm{mod}(i + 1, |N|)$
    $k = k + 1$
  end while
  if $f_{sol} = 1$ **then**
  $\vartheta'' = c$
  $\rho = \lceil (\vartheta'' - \vartheta')/2 \rceil$
  $c = \vartheta' + \rho$
  else
  $\vartheta' = c$
  $\rho = \lceil (\vartheta'' - \vartheta')/2 \rceil$
  $c = \vartheta' + \rho$
  end if
end while
Output: Cycle time ϑ''

Following the construction of π^{ref}, calculate the number m^* workstations with respect to a certain cycle time c by partitioning the sequence into groups, and assigning each group to a workstation such that the total time spent at any workstation does not exceed the cycle time. If this is not possible, then a new workstation is added to the solution and the procedure is repeated. Assume a cycle time c = 11 for the example in Figure 1. Using the initial sequence π^{ref} = {1, 2, 5, 6, 4, 7, 3}, it is possible to group the tasks as (1, 2, 5), (6, 4) and (7, 3), and assign each group to a workstation. In this case, the idle times of each workstation would be 0, 2 and 2, respectively, resulting in a total idle time φ^* = 4. For each assignment, we check whether the zoning constraints (5) and (6) are satisfied. If they are not, then the solution is discarded and the produce repeats until a feasible solution is obtained.

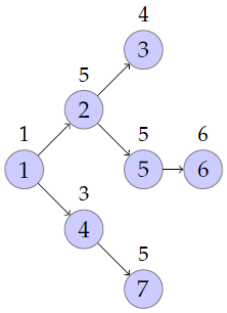

Figure 1. The precedence graph in the example.

Table 1. Zoning restrictions for the instances used in the computational testing

Instance Group	|N|	Zoning restrictions
Lutz2	89	N^- = {(1, 5), (17, 20), (43, 45), (56, 57), (78, 79)} N^+ = {(8, 9), (27, 28), (40, 41), (42, 43}
Warnecke	58	N^- = {(2, 10), (28, 35), (38, 51)} N^+ = {(7, 8), (10, 20), (49, 51}
Wee-Mag	75	N^- = {(6, 7), (12, 21), (34, 41), (53, 61)} N^+ = {(8, 9), (9, 11), (27, 32), (66, 74)}
Mukherje	94	N^- = {(11,14),(26,27),(31,33),(54,56),(76,77)} N^+= {(12,13),(23,24),(33,34),(68,69),(77,78)}
Arcus2	111	N^- = {(5, 6), (17, 19), (30, 31), (53, 54), (63, 65), (90, 92)} N^+ = {(6, 7), (9, 10), (33, 35), (39, 40), (72, 73)}
Barthold	148	N^- = {(1,12),(16,17),(37,38),(47,48),(74,75),(9,90),(118,119),(133,134)} N^+ = {(12, 13), (18, 19), (29, 30), (30, 31), (44, 45), (90, 91), (140, 141)}

3.4. Description of the algorithm

The overall algorithm starts by first producing an upper bound using the RLS, solving the master problem M (P) with an empty set of optimality constraints which yields a solution $\hat{x}$. The solution is then used to solve a subproblem D (α, $\hat{x}$) that identify a workstation s' for which an optimality cut is constructed. The optimality cut along with constraints (14), and combinatorial cuts if active, are all introduced to the M (P) which is resolved to obtain another solution $\hat{x}$. The procedure repeats itself optimality cuts for all workstations are introduced into the master problem.

4. Computational Results

This section presents a computational study to assess the performance of the proposed algorithm and variants in comparison to the models presented earlier. The algorithms are coded in Visual C++, using CPLEX 12.7.1 as the solver, and run on a computer with an Intel Core i5-2450M, a 2.5 GHz CPU and 4GB of memory. The tests are conducted on the various groups of SALBP-II instances available at http://assembly-line-balancing.de/ by including the additional zoning restrictions shown in Table 4, as explained in each of the sections below. For comparison reasons, the solution time for each instance is limited to 10 minutes (600 seconds).

4.1. Comparison of formulations and algorithm variants

As part of the first stage of this experiment, the two models presented previously, namely BM and RC, are compared with the mixed-integer linear optimizer of CPLEX 12.7.1, with the Benders decomposition algorithm of CPLEX 12.7.1 applied to the model RC (denoted ABD), with the exact algorithm described in Section 3 without inequalities (14) and (15), and with the exact algorithm including inequalities (14) (denoted BD+). Table 2 presents the comparison results on 20 instances from the group Lutz2, with the number m of machines that appear next to the instance name ranging from 9 to 28. In the table, the column titled Gap indicates the final optimality gap, expressed in percent, upon termination of the optimization, either upon identifying an optimal solution or upon reaching the time limit, whichever occurs first. The column titled Time indicates the total time required for solving each instance, and it also indicates the best solution identified, which is also the optimal value for instances solved within 600 seconds. Cuts for ABD is an additional column which indicates the number of Benders' optimality cuts applied during the optimization process. The last row in the table indicates the average computation time for the 20 instances.

Table 2. Comparison of five solution methods

Instance(m)	BM			RC			ABD			BD		BD+	
	c^*	Gap	Time	c^*	Gap	Time	c^*	Time	Cuts	c^*	Time	c^*	Time
Lutz2(9)	55	0	4.34	55	0	1.18	177	600.02	14	55	1.2	55	0.88
Lutz2(10)	52	0	24.14	52	0	13.72	177	600.01	13	52	1.9	52	1.54
Lutz2(11)	46	0	2.16	46	0	1.84	166	600.02	26	46	5.2	46	2.97
Lutz2(12)	42	0.79	600.02	42	0	264.45	177	600.02	15	42	4.4	42	2.67
Lutz2(13)	38	0	17.28	38	0	9.7	182	600.02	17	38	4.24	38	3.62
Lutz2(14)	35	0	29	35	0	10.66	178	600.02	22	35	4.18	35	2.5
Lutz2(15)	33	0	20.75	33	0	73.53	133	605.7	62	33	5.88	33	4.39
Lutz2(16)	31	0	220.73	31	0	35.31	127	602.83	119	31	8.19	31	4.82
Lutz2(17)	30	2.96	600.02	30	2.67	600.03	127	600.03	110	30	18.26	30	9.93
Lutz2(18)	28	0.45	600.01	28	0.45	600.03	116	600.01	159	28	20.99	28	11.01
Lutz2(19)	27	4.04	600.05	26	0	590.95	121	606.66	43	26	14.35	26	10.78
Lutz2(20)	25	0	456.41	25	0	273	117	600.01	61	25	12.47	25	8.49
Lutz2(21)	24	1.75	600.03	24	0	271.75	116	600.02	325	24	250.37	24	12.42
Lutz2(22)	24	7.29	601.45	23	3.38	600.05	105	600.02	484	56	600.03	23	12.63
Lutz2(23)	22	1.42	600.02	22	0	432.45	101	612.77	205	22	30.4	22	17.15
Lutz2(24)	21	1.36	600.06	21	0.95	600.05	148	600.02	177	21	354.76	21	80.68
Lutz2(25)	20	2	600.06	20	0	545.08	110	600.02	289	20	144.71	20	30.79
Lutz2(26)	20	5.24	600.03	20	5	600.02	82	600.02	586	20	575.12	20	210.65
Lutz2(27)	19	4.82	600.53	19	3.7	600.03	137	600.01	236	19	134.54	19	29.89
Lutz2(28)	19	8.19	600.77	18	2.38	600.02	82	600.59	148	18	294.37	18	13.7
Average		398.89			336.19			601.44			124.28		23.58

As is evident from Table 2, BD+ is consistently the fastest in solving all the 20 instances optimally, followed by BD, which solves all instances optimally, except for Lutz2(22), for which the optimality gap is 62.5%. As for the formulations, RC is able to identify optimal solutions to 13 of the instances within the time limit, whereas BM optimally solves eight instances, and in larger computational times except for one instance. On the other hand, the ABD is unable to solve any of the instances to optimality within the 600-second time period, producing an optimality gap of 100% for all the instances tested.

4.2. Incorporating the RLS into the BD+

In this section, we report our computational experience with the algorithm BD+ enhanced with the use of the RLS, which we denote by BD++. The main difference here is that BD++ uses, as an upper bound, the value of the best feasible solution obtained with the RLS heuristic. Table 3 shows the results of this experiment on the same instances as in Table 2. For each instance shown in the first column, the

second and third columns show the value of the best solution obtained with a single replication of the RLS and the required solution time, respectively. In this section, we also report the results obtained with a variant of BD++ where the additional inequalities (14) are included in the model prior to the execution of the algorithm, as opposed to being generated on the fly. We denote this variant with BD-, the aim of which is to numerically test the effect of delayed constraint generation on the efficiency of the algorithm. The results of Table 3 indicate that BD++ significantly improves over BD+ in terms of the time required to solve the instances to optimality, yielding an average solution time equal to 4.64 seconds over the 20 instances, as opposed to that of the latter which averages to 23.58 seconds. It also indicates that the savings in computational time by a delayed addition of constraints (14) is significant, which otherwise results in an average computation time of 26.46 seconds, but can be significant at an instance level, e.g., in the case of Lutz2(26), which is solved in 250.65 and 13.13 seconds with BD- and BD++, respectively. Finally, the table shows that the RLS is quick to identify good quality upper bounds for the instances, with the upper bounds deviating by 3.78% from the optimal values on average. Further tests are conducted with RLS, BD- and BD++ on other instances from groups Wee-Mag and Warnecke, which are shown in Table 4, which yield similar findings as regards the comparison of BD- and BD++. In particular, of the 50 instances tested in this table, BD++ optimally solves all but seven of them within the time limit imposed. The RLS, once again, runs in an average of 1.77 seconds, with the upper bounds identified deviating by an average of 3.68% from the optimal values.

Table 3. Comparisons between RLS, BD- and BD++

Instance(m)	RLS		BD-		BD++	
	c^*	Time	c^*	Time	c^*	Time
Lutz2(9)	55	1.71	55	1.25	55	0.88
Lutz2(10)	52	2.12	52	1.03	52	0.82
Lutz2(11)	46	2.23	46	1.1	46	0.66
Lutz2(12)	42	2.11	42	1.93	42	1.88
Lutz2(13)	39	1.97	38	2.04	38	1.34
Lutz2(14)	37	2.22	35	4.54	35	1.53
Lutz2(15)	35	2.14	33	3.48	33	2.48
Lutz2(16)	32	2.45	31	4.13	31	2.06
Lutz2(17)	31	2.13	30	2.61	30	2.43
Lutz2(18)	29	2.12	28	4.88	28	2.39
Lutz2(19)	27	2.71	26	4.03	26	3.97
Lutz2(20)	26	2.65	25	14.04	25	3.27
Lutz2(21)	25	2.32	24	20.48	24	5.96
Lutz2(22)	24	2.15	23	21.61	23	11.71
Lutz2(23)	24	1.87	22	33.03	22	11.2
Lutz2(24)	23	2.82	21	10.46	21	5.16
Lutz2(25)	21	2.98	20	12.68	20	9.61
Lutz2(26)	20	2.76	20	250.65	20	13.13
Lutz2(27)	19	3.14	19	44.58	19	5.53
Lutz2(28)	20	3.17	18	90.66	18	6.79
Average		2.39		26.46		4.64

4.3. Effectiveness of the combinatorial cut generation strategies

The third and last phase of the computational testing numerically evaluates the three cut generation strategies along with effect of the number of additional combinatorial cuts added at each iteration of the algorithm. The tests have been conducted with the BD++ using combinatorial cuts using the cut generation strategies ($\Gamma = 1,2,3$) and where the number ς_C of cuts generated is chosen within $\{2, 5, 10, 25, 50\}$. Once again, these tests use the upper bounds obtained with the RLS. The tests are conducted on the instances that proved difficult to solve in Table 4, namely Wee-Mag(18), Wee-Mag(19), Wee-Mag(20), Warnecke(14), Warnecke(21), Warnecke(22) and Warnecke(25) which were not optimally solved by BD++ within the time limit. The results of these experiments are shown in Table 5, which presents the cycle time for each setting of ς_C and the total computational time under column Time that also includes the execution of the RLS. We refer to this algorithm as BD+CC(ς_C). The results suggest that the performance of the algorithm do not drastically change from one setting to another. On the basis of having solved all the seven instances optimally, we suggest the use of the setting ς_C = 5

which uses strategy Γ_2. Contrasting these results with those presented in Table 4, it is evident that the use of combinatorial cuts is an effective strategy, as it has allowed the solution of instances that otherwise were not solved to optimality within the time limit. Using these settings, we present further results on the performance of the algorithm on other instances in Table 6.

Table 4. Comparisons between RLS, BD- and BD++

Instance(m)	RLS		BD-			BD++		
	c^*	Time	c^*	Gap	Time	c^*	Gap	Time
Warnecke(3)	519	1.65	517	0	0.3	517	0	0.28
Warnecke(4)	388	1.71	388	0	5.13	388	0	3.18
Warnecke(5)	311	1.75	311	0	0.4	311	0	0.16
Warnecke(6)	261	1.28	259	0	0.83	259	0	0.7
Warnecke(7)	224	1.68	222	0	0.91	222	0	0.8
Warnecke(8)	195	1.66	195	0	1.27	195	0	1.09
Warnecke(9)	177	1.16	175	0	1.86	175	0	0.89
Warnecke(10)	162	1.12	159	0	162.6	159	0	109.09
Warnecke(11)	147	1.5	144	0.69	600.18	144	0	32.35
Warnecke(12)	136	1.96	130	0	6.02	130	0	5.25
Warnecke(13)	123	1.34	120	0	9.6	120	0	4.37
Warnecke(14)	114	1.59	114	2.63	600.65	112	0.89	600.07
Warnecke(15)	111	1.22	111	6.31	600.05	106	0	81.57
Warnecke(16)	103	1.75	99	0	10.02	99	0	2.7
Warnecke(17)	101	1.26	101	8.91	600.1	97	0	4.96
Warnecke(18)	94	1.51	94	8.51	600.07	92	0	208.4
Warnecke(19)	89	1.7	89	7.87	600.11	87	0	7.57
Warnecke(20)	85	1.89	85	8.24	600.09	82	0	39.74
Warnecke(21)	81	1.96	81	4.94	600.61	78	1.28	600.1
Warnecke(22)	82	1.55	82	13.41	600.1	74	1.35	600.16
Warnecke(23)	72	1.14	72	4.17	600.06	69	0	72.07
Warnecke(24)	69	1.15	69	4.35	600.18	66	0	43.65
Warnecke(25)	72	1.26	72	9.72	600.13	66	1.51	600.08
Warnecke(26)	73	1.66	65	0	164.22	65	0	127.4
Warnecke(27)	68	1.54	65	0	5.21	65	0	3.81
Warnecke(28)	72	1.85	65	0	3.44	65	0	2.87
Warnecke(29)	68	1.93	65	0	3.15	65	0	2.89
Wee-Mag(3)	504	1.44	500	0	0.69	500	0	0.21
Wee-Mag(4)	380	1.68	375	0	0.21	375	0	0.15
Wee-Mag(5)	305	1.76	300	0	1.89	300	0	0.2
Wee-Mag(6)	261	1.74	250	0	0.55	250	0	0.25
Wee-Mag(7)	224	1.39	215	0	0.91	215	0	0.28
Wee-Mag(8)	195	1.66	188	0	0.89	188	0	0.35
Wee-Mag(9)	173	1.17	167	0	1.41	167	0	0.67
Wee-Mag(10)	152	1.71	150	0	3.07	150	0	1.18
Wee-Mag(11)	140	2.03	140	2.14	600.52	137	0	0.99
Wee-Mag(12)	127	1.28	127	1.57	600.81	125	0	1.81
Wee-Mag(13)	118	2.05	118	1.69	600.43	116	0	1.46

Table 4. Comparisons between RLS, BD- and BD++ (continued)

Instance(m)	RLS		BD-			BD++		
	c^*	Time	c^*	Gap	Time	c^*	Gap	Time
Wee-Mag(14)	112	2.1	112	3.57	600.15	108	0	2.11
Wee-Mag(15)	104	2.82	104	3.85	600.83	100	0	54.21
Wee-Mag(16)	99	2.69	99	5.05	600.77	94	0	2.96
Wee-Mag(17)	93	2.32	93	4.3	600.33	89	0	3.77
Wee-Mag(18)	88	1.95	88	4.55	600.72	87	3.44	600.05
Wee-Mag(19)	90	2.03	90	12.22	600.41	85	7.05	600.54
Wee-Mag(20)	79	2.38	79	5.06	600.13	78	3.84	600.06
Wee-Mag(21)	75	2.77	75	4	600.82	72	0	7.02
Wee-Mag(22)	72	2.8	72	4.17	600.35	69	0	8.2
Wee-Mag(23)	70	2.19	70	4.29	600.79	67	0	568.27
Wee-Mag(24)	72	2.49	72	6.94	600.81	67	0	3.76
Wee-Mag(25)	73	2.45	73	8.22	600.98	67	0	2.17
Average	1.77		331.92			112.34		

Table 5. Results obtained with algorithm BD+CC(ς_C)

Cut Generating Strategy	Instance(m)	ς_C=2	Time	ς_C=5	Time	ς_C=10	Time	ς_C=25	Time	ς_C=50	Time
	Wee-Mag(18)	87	54.61	87	46.67	87	60.65	87	65.66	87	72.15
	Wee-Mag(19)	85	25.91	85	11.25	85	10.78	85	21.91	85	95.76
	Wee-Mag(20)	78	110.76	78	86.15	78	83.81	78	89.45	78	97.15
Γ_1	Warnecke(14)	112	193.37	112	149.86	112	145.45	112	200.21	112	412.89
	Warnecke(21)	78	578.18	78	600.00	78	600.00	78	600.00	78	600.00
	Warnecke(22)	74	300.48	74	312.06	74	405.45	74	411.21	74	431.98
	Warnecke(25)	66	600.00	66	600.00	66	600.00	66	600.00	66	600.00
	Wee-Mag(18)	87	28.76	87	51.12	87	53.14	87	53.45	87	65.10
	Wee-Mag(19)	85	42.22	85	12.45	85	40.13	85	76.45	85	87.54
	Wee-Mag(20)	78	65.65	78	38.15	78	43.76	78	48.23	78	50.43
Γ_2	Warnecke(14)	112	172.65	112	123.12	112	114.51	112	212.21	112	215.21
	Warnecke(21)	78	412.21	78	478.45	78	521.89	78	589.21	78	600.00
	Warnecke(22)	74	267.17	74	215.45	74	413.54	74	452.45	74	327.12
	Warnecke(25)	66	600.00	66	550.45	66	600.00	66	600.00	66	600.00
	Wee-Mag(18)	87	71.87	87	75.09	87	90.13	87	89.99	87	70.21
	Wee-Mag(19)	85	38.58	85	23.45	85	83.15	85	95.21	85	90.14
	Wee-Mag(20)	78	62.01	78	93.12	78	61.34	78	83.32	78	73.76
Γ_3	Warnecke(14)	112	175.71	112	251.12	112	156.65	112	152.12	112	350.43
	Warnecke(21)	78	600.00	78	600.00	78	600.00	78	600.00	78	600.00
	Warnecke(22)	74	234.65	74	218.54	74	385.14	74	465.56	74	401.41
	Warnecke(25)	66	600.00	66	600.00	66	600.00	66	600.00	66	600.00

Table 6. Further results obtained with algorithm BD+CC(5)

Instance(m)	RLS		BD+CC(5)		Instance(m)	RLS		BD+CC(5)	
	c^*	Time	c^*	Time		c^*	Time	c^*	Time
Barthold(4)	1415	1.91	1409	0.23	Mukherje(20)	226	1.57	221	418.2
Barthold(5)	1130	1.87	1127	0.57	Mukherje(21)	216	1.57	208	4.48
Barthold(6)	944	2.17	939	0.67	Mukherje(22)	204	1.61	200	534.29
Barthold(7)	816	2.19	805	1.73	Mukherje(23)	194	1.84	189	598.81
Barthold(8)	717	2.17	705	1.19	Mukherje(24)	184	1.93	179	4.74
Barthold(9)	635	2.25	626	3.04	Mukherje(25)	179	1.82	172	48.44
Barthold(10)	571	2.16	564	1.41	Mukherje(26)	183	1.73	171	3.02
Barthold(11)	518	2.19	513	1.87	Arcus2(3)	50289	1.86	50136	0.19
Barthold(12)	479	2.15	470	3.45	Arcus2(4)	37616	1.74	37605	0.27
Barthold(13)	440	2.86	434	5.31	Arcus2(5)	30096	1.95	30084	0.44
Barthold(14)	407	2.56	403	10.82	Arcus2(6)	25103	1.76	25070	0.5
Barthold(15)	407	2.87	383	2.76	Arcus2(7)	21499	2.01	21489	0.74
Mukherje(4)	1111	1.49	1101	0.12	Arcus2(8)	18828	2.11	18802	6.47
Mukherje(5)	905	1.68	844	0.26	Arcus2(9)	16735	2.43	16713	77.43
Mukherje(6)	729	1.77	704	0.47	Arcus2(10)	15045	1.76	15042	183.94
Mukherje(7)	626	1.63	621	0.64	Arcus2(11)	13682	1.88	13676	93.15
Mukherje(8)	540	1.79	532	0.58	Arcus2(12)	12542	2.64	12536	116.23
Mukherje(9)	484	1.88	477	0.65	Arcus2(13)	11587	2.31	11578	521.69
Mukherje(10)	435	1.7	424	0.82	Arcus2(14)	10758	2.45	10747	562.17
Mukherje(11)	400	1.67	391	1.3	Arcus2(15)	10057	2.56	10031	93.18
Mukherje(12)	366	1.77	358	1.39	Arcus2(16)	9444	2.31	9432	513.91
Mukherje(13)	334	1.63	325	31.32	Arcus2(17)	8405	2.23	8374	510.72
Mukherje(14)	320	1.76	311	2.5	Arcus2(18)	8405	2.56	7939	218.44
Mukherje(15)	296	1.74	288	2.4	Arcus2(19)	7979	2.79	7528	310.01
Mukherje(16)	272	1.66	268	111.34	Arcus2(20)	7544	2.88	7195	544.34
Mukherje(17)	260	1.79	251	3.09	Arcus2(21)	7224	2.93	6863	595.58
Mukherje(18)	246	1.61	239	4.3	Arcus2(22)	6864	2.67	6594	166.86
Mukherje(19)	234	1.65	226	2.53	Arcus2(23)	6605	2.83	6468	5.91

4.4. Results on the simple assembly line balancing problem

As the problem studied in this paper extends SALBP-II, we present, in this section, some preliminary results on the performance of the algorithm variants when applied to the latter. The SALBP-II can be solved using highly efficient bespoke algorithms, such as the branch-and-bound algorithm SALOME (Scholl & Klein, 1997), which can be accessed by visiting https://assembly-line-balancing.de/salbp/salome/. While the purpose of the exact algorithm described in this paper is not to solve the SALBP-II, we provide some comparisons with SALOME in order to provide an overview of the capabilities that this new algorithm might provide. The experiments were conducted on the same set of instances shown in Table 4, where SALOME was compared with BD+ and BD+CC(5). In Table 7, the results are presented using the same format as the previous tables, except for the numbers in parentheses, which represent the best lower bounds obtained in the event that the corresponding instance was not optimally solved within the 600-second time limit. Due to the use of combinatorial cuts within that algorithm, the lower bounds for BD+CC(5) are not reported. According to the comparisons presented in Table 7, SALOME, as a tailored algorithm for solving the SALBP-II, produces optimal solutions in a very short timeframe for all Warnecke instances. Additionally, BD+CC(5) is also capable of identifying optimal solutions to the same instances, but the computation time is considerably longer. It is to be expected, particularly since the latter algorithm involves an iterative solution, which is time consuming. SALOME is, however, unable to guarantee optimality for the Wee-Mag group within the time period in some instances. The Wee-Mag(10), Wee-Mag(14), Wee-Mag(16), Wee-Mag(21), Wee-Mag(22), Wee-Mag(24), Wee-Mag(25) and Wee-Mag(27) instances can be optimally solved, either by the BD+ or BD+CC(5), in much shorter computational times. However, despite the results confirming the general superiority of SALOME as a bespoke solution tool for the SALBP-II, they also suggest that the proposed algorithm and variants may be considered as an alternative solution method for cases that proved difficult to resolve using SALOME.

Table 7. Comparisons between SALOME, BD+ and BD++ on SALBP-II

Instance(m)	SALOME		BD+		BD+CC(5)	
	c^*	Time	c^*	Time	c^*	Time
Warnecke(3)	516	0.025	516	0.12	516	0.34
Warnecke(4)	387	0.004	387	1.39	387	0.7
Warnecke(5)	310	0.004	310	0.34	310	0.65
Warnecke(6)	258	0.004	258	1.15	258	2.49
Warnecke(7)	222	0.006	222	1.28	222	0.87
Warnecke(8)	194	0.006	194	1.26	194	3.39
Warnecke(9)	172	0.005	172	6.35	172	8.54
Warnecke(10)	155	0.007	155	3.34	155	10.74
Warnecke(11)	142	0.008	142	4.84	142	22.95
Warnecke(12)	130	0.01	130	6.19	130	21.47
Warnecke(13)	120	0.008	120	48.85	120	11.92
Warnecke(14)	111	0.008	111	227.7	111	17.83
Warnecke(15)	104	0.012	104	132.2	104	8.05
Warnecke(16)	98	0.039	98	13.04	98	18.13
Warnecke(17)	92	0.025	92	21.05	92	17.05
Warnecke(18)	87	0.788	87	213.5	87	200.42
Warnecke(19)	84	0.031	84	560.7	84	541.41
Warnecke(20)	79	0.079	79	226.5	79	32.03
Warnecke(21)	76	0.23	76 (75)	600.2	76	374.16
Warnecke(22)	73	2.511	73 (72)	600.3	73	37.08
Warnecke(23)	69	0.165	69	290.7	69	145.12
Warnecke(24)	66	1.334	66	152.6	66	18.94
Warnecke(25)	64	1.15	65 (64)	600.3	64	155.12
Warnecke(26)	64	2.551	64	55.91	64	13.21
Warnecke(27)	60	0.136	60 (59)	600.1	60	36.15
Warnecke(28)	59	0.112	59 (57)	600	59	272.78
Warnecke(29)	56	0.486	57 (55)	600.1	56	345.95
Wee-Mag(3)	500	0.037	500	0.14	500	0.16
Wee-Mag(4)	375	0.005	375	0.37	375	1.13
Wee-Mag(5)	300	0.007	300	0.4	300	0.98
Wee-Mag(6)	250	0.007	250	0.97	250	0.78
Wee-Mag(7)	215	0.019	215	0.37	215	0.41
Wee-Mag(8)	188	0.006	188	1.28	188	0.84
Wee-Mag(9)	167	0.055	167	1.91	167	1.13
Wee-Mag(10)	151 (150)	600.45	150	3.13	150	1.51
Wee-Mag(11)	137	0.013	137	1.78	137	1.09
Wee-Mag(12)	125	0.012	125	2.92	125	5.15
Wee-Mag(13)	116	0.011	116	2.4	116	1.51
Wee-Mag(14)	109 (108)	600.32	108	7.45	108	28.66
Wee-Mag(15)	100	321.09	100	45.2	100	112.45
Wee-Mag(16)	96 (94)	600.21	94	9	94	2.04
Wee-Mag(17)	89	0.01	89	33.5	89	37.36
Wee-Mag(18)	88 (84)	600.05	87 (85)	600.2	87	600.12
Wee-Mag(19)	90 (79)	600.03	85 (80)	600.3	85	600.05
Wee-Mag(20)	77	0.006	77	204	77	15.23
Wee-Mag(21)	73 (72)	600.31	72	13.6	72	16.5
Wee-Mag(22)	70 (69)	600.45	69	30.59	69	6.6
Wee-Mag(23)	69 (66)	600.06	67 (66)	600.1	67	600.07
Wee-Mag(24)	67 (66)	600.01	66	32.2	66	12.34
Wee-Mag(25)	66 (65)	600.12	65	127.5	65	55.12
Wee-Mag(26)	65 (64)	600.03	65 (63)	600.1	65	600.36
Wee-Mag(27)	67 (62)	600.05	65	120.8	65	98.12
Wee-Mag(28)	64 (63)	600.04	64 (58)	600	64	600.04
Wee-Mag(29)	63	0.007	63	194.4	63	25.41
Wee-Mag(30)	56	0.005	56	78.74	56	9.41

5. Discussion and Conclusions

This paper describes an algorithm for the type II simple assembly line balancing problem with zoning constraints, which, to the best of the authors' knowledge, is the first exact algorithm to be available to solve this problem. Essentially, the algorithm solves a formulation of the problem where some of the constraints are initially relaxed and added subsequently on the fly iteratively, in the spirit of Benders decomposition. The algorithm has demonstrated a promising computational performance by

incorporating additional features such as a local search heuristic to provide good quality upper bounds and combinatorial cuts to discard feasible solutions. Particularly, all the instances tested here have been optimally resolved within a time limit of 10 minutes. Additionally, the algorithm and its variants are capable of solving the type-II simple assembly line balancing problem and have shown promising results in some instances that are difficult to solve with conventional methods.

References

Akpınar, S., & Bayhan, G. M. (2011). A hybrid genetic algorithm for mixed model assembly line balancing problem with parallel workstations and zoning constraints. *Engineering Applications of Artificial Intelligence,* 24(3), 449-457. doi: 10.1016/j.engappai.2010.08.006

Akpinar, S., Elmi, A., & Bektaş, T. (2017). Combinatorial Benders cuts for assembly line balancing problems with setups. *European Journal of Operational Research,* 259(2), 527-537. doi: 10.1016/j.ejor.2016.11.001

Amen, M. (2006). Cost-oriented assembly line balancing: Model formulations, solution difficulty, upper and lower bounds. *European Journal of Operational Research,* 168(3), 747-770. doi: 10.1016/j.ejor.2004.07.026

Balas, E. (1965). An additive algorithm for solving linear programs with zero-one variables. *Operations Research,* 13(4), 517-546. doi: 10.1287/opre.13.4.517

Baybars, İ. (1986). A survey of exact algorithms for the simple assembly line balancing problem. *Management Science,* 32(8), 909-932. doi: 10.1287/mnsc.32.8.909

Baykasoglu, A., & Dereli, T. (2008). Two-sided assembly line balancing using an ant-colony based heuristic. *The International Journal of Advanced Manufacturing Technology,* 36(5-6), 582-588. doi: 10.1007/s00170-006-0861-3

Benders, J. F. (1962). Partitioning procedures for solving mixed-variables programming problems. *Numerische Mathematik*, 4(1), 238-252.

Bowman, E. H. (1960). Assembly-line balancing by linear programming. *Operations Research,* 8(3), 385-389. doi: 10.1287/opre.8.3.385

Boysen, N., Fliedner, M., & Scholl, A. (2008). Assembly line balancing: Which model to use when? *International Journal of Production Economics,* 111(2), 509-528. doi: 10.1016/j.ijpe.2007.02.026

Codato, G., & Fischetti, M. (2006). Combinatorial Benders' cuts for mixed-integer linear programming. *Operations Research,* 54(4), 756-766. doi: 10.1287/opre.1060.0286

Côté, J.-F., Dell'Amico, M., & Iori, M. (2014). Combinatorial Benders' cuts for the strip packing problem. *Operations Research,* 62(3), 643-661. doi: 10.1287/opre.2013.1248

Deng, G., & Gu, X. (2012). A hybrid discrete differential evolution algorithm for the no-idle permutation flow shop scheduling problem with makespan criterion. *Computers & Operations Research,* 39(9), 2152-2160. doi: 10.1016/j.cor.2011.10.024

Furugi, A. (2022). Sequence-dependent time- and cost-oriented assembly line balancing problems: a combinatorial Benders' decomposition approach. *Engineering Optimization*, 54(1), 170-184. doi: 10.1080/0305215X.2021.1953003

Gokcen, H., & Erel, E. (1997). A goal programming approach to mixed-model assembly line balancing problem. *International Journal of Production Economics,* 48(2), 177-185. doi: 10.1016/S0925-5273(96)00069-2

Hamzadayı, A. (2018). Balancing of mixed-model two-sided assembly lines using teaching-learning based optimization algorithm. *Pamukkale University Journal of Engineering Sciences*, 24(4), 682-691. doi: 10.5505/pajes.2017.14227

Hazır, Ö., & Dolgui, A. (2013). Assembly line balancing under uncertainty: Robust optimization models and exact solution method. *Computers & Industrial Engineering,* 65(2), 261-267. doi: 10.1016/j.cie.2013.03.004

Hazır, Ö., & Dolgui, A. (2015). A decomposition based solution algorithm for u-type assembly line balancing with interval data. *Computers & Operations Research,* 59, 126-131. doi: 10.1016/j.cor.2015.01.010

Held, M., Karp, R. M., & Shareshian, R. (1963). Assembly-line balancing - dynamic programming with precedence constraints. *Operations Research,* 11(3), 442-459. doi: 10.1287/opre.11.3.442

Hoffmann, T. R. (1992). Eureka: A hybrid system for assembly line balancing. *Management Science,* 38(1), 39-47. doi: 10.1287/mnsc.38.1.39

Holland, J. H. (1975). *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence*. University of Michigan Press, Ann Arbor, MI.

Huang, D., Mao, Z., Fang, K. & Yuan, B. (2021). Combinatorial Benders decomposition for mixed-model two-sided assembly line balancing problem. *International Journal of Production Research*, 60(8), 2598-2624. doi: 10.1080/00207543.2021.1901152

Jackson, J. R. (1956). A computing procedure for a line balancing problem. *Management Science,* 2(3), 261-271. doi: 10.1287/mnsc.2.3.261

Johnson, R. V. (1988). Optimally balancing large assembly lines with FABLE. *Management Science,* 34(2), 240-253. doi: 10.1287/mnsc.34.2.240

Karp, R. M. (1972). *Reducibility among combinatorial problems*. In Complexity of Computer Computations, 85-103. Springer.

Klein, M. (1963). On assembly line balancing. *Operations Research,* 11(2), 274-281. doi: 10.1287/opre.11.2.274

Klein, R., & Scholl, A. (1996). Maximizing the production rate in simple assembly line balancing–a branch and bound procedure. *European Journal of Operational Research,* 91(2), 367-385. doi: 10.1016/0377-2217(95)00047-X

Özbakır, L., & Tapkan, P. (2011). Bee colony intelligence in zone constrained two-sided assembly line balancing problem. *Expert Systems with Applications,* 38(9), 11947-11957. doi: 10.1016/j.eswa.2011.03.089

Özcan, U., & Toklu, B. (2009). Multiple-criteria decision-making in two-sided assembly line balancing: A goal programming and a fuzzy goal programming models. *Computers & Operations Research,* 36(6), 1955-1965. doi: 10.1016/j.cor.2008.06.009

Pan, Q.-K., & Ruiz, R. (2014). An effective iterated greedy algorithm for the mixed no-idle permutation flowshop scheduling problem. *Omega,* 44, 41-50. doi: 10.1016/j.omega.2013.10.002

Pastor, R., & Ferrer, L. (2009). An improved mathematical program to solve the simple assembly line balancing problem. *International Journal of Production Research,* 47(11), 2943-2959. doi: 10.1080/00207540701713832

Patterson, J. H., & Albracht, J. J. (1975). Assembly-line balancing: zero-one programming with fibonacci search. *Operations Research,* 23(1), 166-172. doi: 10.1287/opre.23.1.166

Ritt, M., & Costa, A. M. (2018). Improved integer programming models for simple assembly line balancing and related problems. *International Transactions in Operational Research,* 25(4), 1345-1359. doi: 10.1111/itor.12206

Scholl, A., & Becker, C. (2006). State-of-the-art exact and heuristic solution procedures for simple assembly line balancing. *European Journal of Operational Research,* 168(3), 666-693. doi: 10.1016/j.ejor.2004.07.022

Scholl, A., & Klein, R. (1997). SALOME: A bidirectional branch-and-bound procedure for assembly line balancing. *INFORMS Journal on Computing,* 9(4), 319-334. doi: 10.1287/ijoc.9.4.319

Schrage, L., & Baker, K. R. (1978). Dynamic programming solution of sequencing problems with precedence constraints. *Operations Research,* 26(3), 444-449. doi: 10.1287/opre.26.3.444

Talbot, F. B., & Patterson, J. H. (1984). An integer programming algorithm with network cuts for solving the assembly line balancing problem. *Management Science,* 30(1), 85-99. doi: 10.1287/mnsc.30.1.85

Vilarinho, P. M., & Simaria, A. S. (2002). A two-stage heuristic method for balancing mixed model assembly lines with parallel workstations. *International Journal of Production Research,* 40(6), 1405-1420. doi: 10.1080/00207540110116273

White, W. W. (1961). Letter to the editor – comments on a paper by Bowman. *Operations Research,* 9(2), 274–276. doi: 10.1287/opre.9.2.274