A fast branch, bound and remember algorithm for disassembly line balancing problem

Zixiang Li [ab], Zeynel Abidin Çil [c*], Süleyman Mete [d], Ibrahim Kucukkoc [e]

[a]Key Laboratory of Metallurgical Equipment and Control Technology, Wuhan University of Science and Technology, Wuhan, People's Republic of China
[b]Hubei Key Laboratory of Mechanical Transmission and Manufacturing Engineering, Wuhan University of Science and Technology, Wuhan, People's Republic of China
[c]Department of Industrial Engineering, Izmir Democracy University, Izmir, Turkey
[d]Department of Industrial Engineering, Munzur University, Tunceli, Turkey; [e]Industrial Engineering Department, Balikesir University, Balikesir, Turkey

Abstract

In recent years, the interests of disassembly line have increased owing to economic reasons and the increase of environmental awareness. Effective line can provide many advantages in terms of economic aspect and it facilitates competition the companies with others. This study contributes to the relevant literature by a branch, bound and remember algorithm for disassembly line balancing problem with AND/OR precedence. The proposed exact solution method employs the memory-based dominance rule to eliminate the reduplicated sub-problems by storing all the searched sub-problems and to utilise cyclic best-first search strategy to obtain high-quality complete solutions fast. In this paper, minimising the number of stations is taken as the performance measure. The proposed methodology is tested on a set of 260 instances and compared with the mathematical model using CPLEX solver and five well-known metaheuristics. Computational results show that the proposed method is capable of obtaining the optimal solutions for all the tested instances with less than 0.1 seconds on average. Additionally, comparative study demonstrates that the proposed method is the state-of-the-art algorithm and outperforms the CPLEX solver and metaheuristics in terms of both solution quality and search speed aspects.

Keywords: combinatorial optimisation, branch and bound remember, disassembly, line balancing, exact solution algorithm

Corresponding author: Zeynel Abidin Çil, cilzeynelabidin@gmail.com

A fast branch, bound and remember algorithm for disassembly line balancing problem

Zixiang Li[1,2], Zeynel Abidin Çil[3*], Süleyman Mete[4], Ibrahim Kucukkoc[5]

[1]Key Laboratory of Metallurgical Equipment and Control Technology, Wuhan University of Science and Technology, Wuhan, Hubei, China
[2]Hubei Key Laboratory of Mechanical Transmission and Manufacturing Engineering, Wuhan University of Science and Technology, Wuhan, Hubei, China
[3]Department of Industrial Engineering, Izmir Democracy University, Izmir, Turkey
[4]Department of Industrial Engineering, Munzur University, Tunceli, Turkey
[5]Industrial Engineering Department, Balikesir University, Cagis Campus, Balikesir 10145, Turkey.

Email: zixiangliwust@gmail.com (Z. Li); cilzeynelabidin@gmail.com (Z. A. Çil); suleymanmete@munzur.edu.tr (S. Mete); ikucukkoc@balikesir.edu.tr (I. Kucukkoc)

*Corresponding author
E-mail:cilzeynelabidin@gmail.com
Phn:+905055590935

A fast branch, bound and remember algorithm for disassembly line balancing problem

Abstract: In recent years, the interests of disassembly line have increased owing to economic reasons and the increase of environmental awareness. Effective line can provide many advantages in terms of economic aspect and it facilitates competition the companies with others. This study contributes to the relevant literature by a branch, bound and remember algorithm for disassembly line balancing problem with AND/OR precedence. The proposed exact solution method employs the memory-based dominance rule to eliminate the reduplicated sub-problems by storing all the searched sub-problems and to utilize cyclic best-first search strategy to obtain high-quality complete solutions fast. In this paper, minimizing the number of stations is taken as the performance measure. The proposed methodology is tested on a set of 260 instances and compared with the mathematical model using CPLEX solver and five well-known metaheuristics. Computational results show that the proposed method is capable of obtaining the optimal solutions for all the tested instances with less than 0.1 seconds on average. Additionally, comparative study demonstrates that the proposed method is state of the art algorithm and outperforms the CPLEX solver and metaheuristics in terms of both solution quality and search speed aspects.

Keywords: Combinatorial optimization; Branch and bound remember; Disassembly; Line balancing; Exact solution algorithm.

1. Introduction

Minimizing the environmental impact of waste materials is one of the most significant issues in today's economic and environmental conditions. Product recovery has been vital and popular in industry for social, environmental and economic benefits due to rigid environmental regulations (Güngör and Gupta, 1999; Mete et al., 2018). Product recovery aims at reducing the environmental pollution and minimizing the total waste by remanufacturing or recycling the valuable component of the product. During product recovery, the product disassembly is the first and essential step, where the valuable components are dismantled from the discarded products through a set of disassembly operations.

The disassembly operations are generally performed on a line of disassembly that contains of serial stations. Disassembly lines are widely applied to disassemble the discarded products because of its high productivity and suitability for automation. While there are a number of sub-problems within the disassembly line, line balancing is at the forefront. In addition, the disassembly line balancing problem (DLBP) emerges and attract increasing attention to optimize the line efficiency or other optimization criteria (Güngör and Gupta, 2002). DLBP without loss of generality, can be described as portioning and assigning a set of disassembly tasks to stations with one or several optimization criteria, while satisfying the precedence constraint and cycle time constraint. Hence, the efficiently designed and balanced disassembly line has a remarkable industrial and environmental importance.

As far as precedence relation is concerned, the precedence relation of the DLBP is much more complex than the assembly line balancing problem (ALBP). ALBP contains functional and physical precedence constraints while DLBP contains generally physical constraints. Assembly line process contains a functional finish product, so precedence relation can be improved according to functional and physical constraints in ALBP. However, just physical constraint is considered in precedence relations of DLBP (Mete et al. 2016a; Li and Boucher, 2017). There is mainly AND precedence relationship in ALBP; the precedence relation of the DLBP includes AND precedence, OR precedence, complex AND/OR precedence relationships (Güngör and Gupta, 2002). There are also other studies on OR successor (Koc et al., 2009), where only one of the successors is selected. This study mainly focuses on the DLBP without OR successor following Güngör and Gupta (2002); the problems without OR successor have great applications in real industry and have attracted many attentions (Ding et al., 2010; Kalayci and Gupta, 2013b, 2013c; Ren et al., 2017; Ren et al., 2018a).

DLBP was first introduced by Güngör and Gupta (1999), where a systematic approach-oriented heuristic method is presented. Since then, much attention has been paid to this active research area. There exist heuristic approaches and exact solution methods to solve the problem in the literature. Firstly, heuristic-based solution approach studies are examined, and then mathematical modeling solution approaches are analyzed in the paragraph. Gungor and Gupta (2001) employed the shortest-path formulation to tackle the DLBP in the presence of task failures and Güngör and Gupta (2002) formulated the DLBPs in different situations and presented a heuristic method. McGovern and Gupta (2003) presented a greedy/2-opt hybrid algorithm to solve the multi-objective DLBP. Following this, Ren et al. (2018b) extended this method to multi-objective DLBP with weights-based multi-criteria decision. Besides, Mete et al.(2016a) developed another heuristic method named beam search in order to minimize the station number. Due to the NP-hard nature of DLBP proved by McGovern and Gupta (2007) metaheuristic methods become popular to obtain high-quality solutions in acceptable computational time. McGovern and Gupta (2007) developed a genetic algorithm for DLBP, and later Kalayci, Polat, and Gupta (2016) extended this method to sequence-dependent DLBP. Other applied algorithms included the ant colony optimization algorithms (Agrawal and Tiwari, 2008; Ding et al., 2010; Kalayci and Gupta, 2013a), tabu search algorithm (Kalayci and Gupta, 2014), particle swarm optimization (Kalayci and Gupta, 2013c; Xiao et al.,2017), gravitational search algorithm (Ren et al., 2017), artificial bee colony (Kalayci and Gupta, 2013b; Kalayci et al., 2015; Liu and Wang, 2017)artificial fish swarm algorithm (Zhang et al., 2017), discrete bees algorithm (Liu et al., 2018), variable neighborhood search algorithm (Ren et al., 2018a) and firefly algorithm (Zhu, Zhang and Wang, 2018). Apart from the aforementioned heuristics/metaheuristics, there are some other mathematical programming techniques attempting to solve DLBP optimally. Altekin, Kandiller, and Ozdemirel (2008) employed mixed integer programming (MIP) formulation for profit-oriented DLBP. Koc, Sabuncuoglu, and Erel (2009) proposed one MIP model for DLBP using an AND/OR graph. Other exact techniques were developed for task-failure-driven rebalancing of disassembly lines (Altekin and Akkan, 2012), resource constrained DLBP (Mete et al. 2016b), piecewise linear approximation (Altekin, 2017), linear physical programming (Ilgın et al., 2017). Recently, Li et al. (2019) proposed branch, bound and remember (BBR) algorithm for simple DLBP using an AND/OR graph. The main differences between the study of Li et al. (2019) and current paper is structure of relations which are AND/OR successors and AND/OR precedence. These relations transform the problem into different problem. Therefore, different lower bound, upper bound or algorithm structure can necessary to solve the problem effectively. A novel mathematical model for joint design assembly and disassembly line balancing problem was introduced by Mete et al.

(2018). Nevertheless, they might be impractical in solving large-size practical problems due to high computation time (Özceylan et al., 2018). A summary of the relevant DLB literature is given in Table 1, differentiated with respect to performance measures and solution approaches.

Table 1: Summary of the literature on DLBP

Papers (sorted by year)	**Line Type**	**Structure of parameters**	**Disassembly Level**	**Product Type**	**Solution Approaches**
Gungor and Gupta (2001)	Straight	Deterministic	Completely	Single	Heuristic
McGovern and Gupta (2007)	Straight	Deterministic	Completely	Single	Genetic algorithm
Agrawal and Tiwari (2008)	U-type	Stochastic	Completely	Mixed	Ant colony algorithm
Altekin, Kandiller, and Ozdemirel (2008)	Straight	Deterministic	Partially	Single	Mixed integer linear programming
Koc, Sabuncuoglu, and Erel (2009	Straight	Deterministic	Completely	Single	Mixed integer linear programming
Ding et al. (2010)	Straight	Deterministic	Completely	Single	Ant colony algorithm
Altekin and Akkan (2012)	Straight	Deterministic	Partially	Single	Mixed integer linear programming
Kalayci and Gupta (2013a)	Straight	Deterministic	Completely	Single	Ant colony algorithm
Kalayci and Gupta (2013b)	Straight	Deterministic	Completely	Single	Artificial bee colony
Kalayci and Gupta (2013c)	Straight	Deterministic	Completely	Single	Particle swarm optimization
Paksoy et al. (2013)	Straight	Fuzzy	Completely	Mixed	0-1 fuzzy goal programming
Aydemir-Karadag and Turkbey (2013)	Parallel	Stochastic	Completely	Single	Genetic algorithm
Kalayci and Gupta (2014)	Straight	Deterministic	Completely	Single	Tabu search algorithm
Avikal, Mishra, and Jain (2014)	Straight	Fuzzy	Completely	Single	Fuzzy AHP, Kano model M-TOPSIS, Heuristic
Bentaha, Battaïa, and Dolgui (2014)	Straight	Stochastic	Completely	Single	Monte Carlo sampling technique, L-shaped algorithm
Kalayci et al. (2015)	Straight	Fuzzy	Completely	Single	Hybrid discrete artificial bee colony algorithm
Hezer and Kara (2015)	Parallel	Deterministic	Completely	Single	Network-based shortest route model
Bentaha, Battaïa, and Dolgui (2015)	Straight	Stochastic	Completely	Single	Exact solution
Kalayci, Polat and Gupta (2016)	Straight	Deterministic	Completely	Single	Hybrid genetic algorithm
Mete et al. (2016a)	Straight	Deterministic	Completely	Single	Beam search algorithm
Ilgın, Akçay and Araz (2017)	Straight	Deterministic	Completely	Mixed	Linear physical programming
Altekin, (2017)	Straight	Stochastic	Partially	Single	
Ren et al. (2017)	Straight	Deterministic	Partially	Single	Gravitational search algorithm
Liu and Wang, (2017)	Straight	Deterministic	Completely	Single	Artificial bee colony
Zhang et al. (2017)	Straight	Deterministic	Completely	Single	Artificial fish swarm algorithm, Fuzzy programming
Ren et al. (2018b)	Straight	Fuzzy	Completely	Single	2-opt algorithm
Mete et al. (2018)	Parallel	Deterministic	Completely	Single	Mixed integer linear programming; ant colony optimisation
Zhu, Zhang and Wang (2018)	Straight	Deterministic	Completely	Single	Pareto firefly algorithm
Li et al. (2019)	Straight	Deterministic	Completely	Single	Exact and heuristic algorithms
Mete et al. (2019)	Straight	Deterministic	Completely	Single	Mixed integer programming
Proposed Study	Straight	Deterministic	Completely	Single	BBR algorithm

In the light of aforementioned literature and a recent review paper by Özceylan et al. (2018), it is observed that the current exact MIP methods might suffer from tremendous computation time in solving large-size practical problems. Thus, this research makes the first attempt to solve the DLBP with branch, bound and remember (BBR) algorithm, which is capable of tackling both small-size and large-size instances optimally. BBR is selected as it has achieved the state-of-the-art results for ALBP (Li, Kucukkoc, and Zhang, 2018; Morrison, Sewell, and Jacobson, 2014; Sewell and Jacobson, 2012), and it is capable of solving the ALBPs with up to 1,000 tasks. The proposed BBR utilizes the memory-based dominance rule to eliminate the reduplicated sub-problems and develops the cyclic best-first search strategy with proper station load selection criterion to obtain high-quality complete solutions fast. A comprehensive study on a set of 260 instances is carried out to test the performance of the proposed BBR, and results of the BBR method are also compared with that by CPLEX solver and five well-known metaheuristics. Computational results demonstrate the superiority of the BBR method in both solution quality and search speed. Moreover, the BBR method obtains optimal solutions for all the tested instances within 0.1 second on average.

The rest of the paper is organized as follows. Section 2 describes the solved problem and presents the detailed mathematical model. The proposed BBR methodology is explained in detail in Section 3. Section 4 presents the computational study to test the performance of the proposed method. Conclusions, directions for future research and managerial impacts are argued in the last section.

2. Problem formulation

This part firstly introduces the main features of the considered DLBP with an example, and later presents the detailed mathematical model.

2.1 Problem description

DLBP assigns a set of tasks to each station for each product to be disassembled by considering cycle time constraints and precedence relationships. In this paper, minimizing the number of stations is taken as the performance measure. In the DLBP literature, there are various precedence relations such as AND precedence, OR precedence, complex AND/OR precedence, and OR successor. This paper mainly centers on the AND/OR precedence relations. Figure 1 illustrates an example precedence diagram with 12 disassembly tasks taken from Ren et al. (2017). In this figure, a disassembly task is represented by a node i ($i = 1,2,\cdots,12$). Furthermore, two dummy tasks, A1 and A2, are utilized to describe the complex AND/OR precedence relations. There is a positive operation time t_i for operating a real task; there is no operation time for a dummy task or the operation time of a dummy task is set to zero. There are two main constraints in DLBP: cycle time constraint and precedence constraint. Cycle time constraint indicates that the total operation time of the disassembly tasks on a station is less than or equal to a given cycle time. The precedence constraint in DLBP is much more complex than that of ALBP. Specifically, in the AND precedence, the predecessors of a task, referred to as AND predecessors, must be completed before operating this task, e.g. task 5 can be operated only when both task 3 and task 4 have been completed. In the OR precedence, at least one of the predecessors of a task, referred to as OR predecessor, must be completed before operating this task. In this figure, the OR predecessors are linked with an arc, e.g. task 1 and task 2 are the OR predecessors of task 4. In other words, task 4 can be operated when either task 1 or task 2 is completed. Likewise, the OR relations can be predecessors as well successors. OR successors relations indicate that at most one of the tasks in a specified set can be performed after one task completed. This relation type is presented by Altekin et al. (2008). There are also more complex precedence relations as presented in this figure, AND within OR and OR within AND, where dummy tasks are employed to describe these two complex precedence relations clearly. In this type of priority relation, the priority relation is first observed AND then OR. According to Figure 1, before performing task 8, task 5 and task6 or task 11 must be done. Hence, task 5-task 6 and task 11 have connected each other with AND relation, so two group of tasks are connected task 8 with OR type relations. As can be understood from the above precedence and successors relations, there is no only one way, such as ALBPs, for the disassembly of a product.

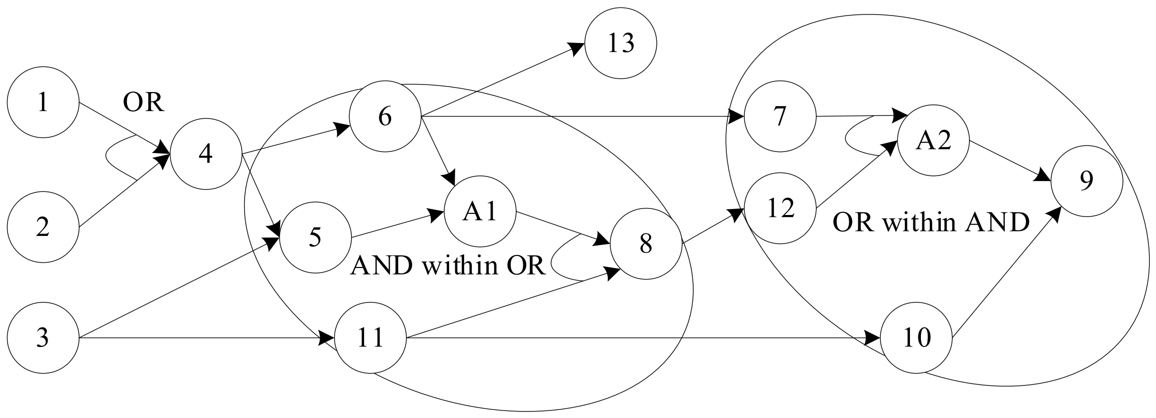

Fig. 1 Precedence diagram of a product

2.2 Mathematical model

On the basis of Altekin and Akkan (2012) and Kalaycılar, Azizoğlu, and Yeralan (2016), the mathematical model is formulated as follows. The utilized notations are first introduced as follows.

Indices:

i, j	Task/part index, $i,j \in \{1,2,\cdots,N\}$, where N is the number of real and dummy tasks/parts.
m,n	Station index, $m,n \in \{1,2,\cdots,M\}$, where M is the maximum number of stations allowed to be opened.
CT	Given cycle time.
t_i	Operation time of performing task i.
$\mathrm{ANDP}(i)$	Set of AND predecessor of task i.
$\mathrm{ORP}(i)$	Set of OR predecessor of task i.
ORPT	Set of tasks which have OR predecessors.

Decision variable:

x_{im}	1, if task i is allocated to station m; 0, otherwise.

Auxiliary variable

y_m	1, if station m is opened; 0, otherwise.

In the considered DLBP, cycle time constraint and precedence constraint must be satisfied. The detailed mathematical formulation is provided using constraints (1-6). It is worth to note that a dummy task is necessary when utilizing this model to solve the DLBP with the complex AND within OR precedence relations. However, it is acceptable to use one dummy task or not to solve the DLBP with OR within AND precedence relations.

$$\text{Min } f = \sum_{m=1}^{M} y_m \tag{1}$$

$$\sum_{i=1}^{N} t_i \cdot x_{im} \le y_m.CT \ \ \forall m \tag{2}$$

$$\sum_{m=1}^{M} x_{im} = 1 \ \ \forall i \tag{3}$$

$$x_{im} \le \sum_{n=1}^{m} x_{jn} \ \ \forall m, i, \text{and } j \in \text{ANDP}(i) \tag{4}$$

$$x_{im} \le \sum_{j \in \text{ORP}(i)} \sum_{n=1}^{m} x_{jn} \ \ \forall m, i \in \text{ORPT} \tag{5}$$

$$x_{im}, y_m \in \{0, 1\} \ \ \forall i, m \tag{6}$$

Specifically, the objective function (1) minimizes the number of opened stations. Constraints (2) deal with the cycle time constraint, ensuring that the total operation time on any station is less than or equal to the given cycle time. Constraints (3) indicate that one task must be allocated to one station. Constraints (4) and constraints (5) tackle the precedence relations, where Constraints (4) handle AND precedence relations and constraints (5) handle OR precedence relations. Constraints (4) indicate that a task can be allocated only when all its AND predecessors have been allocated to the former station or the same station. Constraints (5) guarantee that a task can be allocated when at least one of its OR predecessors has been allocated to the former station or the same station. Constraints (6) mean that the decision variable and the auxiliary variable take 0 or 1. The above mathematical model is a linear mixed integer-programming model, which could be capable of solving using CPLEX solver.

3. Proposed branch, bound and remember algorithm

As one of the exact methods, BBR algorithm (Borba, Ritt, and Miralles, 2018; Li, 2017; Li et al., 2017; Li et al., 2018; Morrison et al., 2014; Sewell and Jacobson, 2012; Vilà and Pereira,

2014) has produced promising results in ALBPs (Battaïa and Dolgui, 2013). For instance, BBR achieves the state-of-the-art results for simple ALBP (Morrison et al., 2014; Sewell and Jacobson, 2012) and U-shaped ALBP (Li et al., 2018). The main feature of BBR method is that it stores all the searched sub-problems and uses memory to eliminate redundant states. Nevertheless, from a recent review paper by Özceylan et al. (2018), and to the authors' best knowledge, there exists no application of exact method in solving DLBPs for OR predecessors. Hence, this research put forward the first attempt to solve DLBP with BBR methods. The main procedure and the segments are given in the following sub-sections.

3.1 Main procedure of the BBR method

The main procedure of the proposed BBR method is illustrated in Algorithm 1 as follows. BBR comprises of three phases: Phase I aims at obtaining a high-quality upper bound (UB), Phase II tries to find new UB utilizing cyclic best-first search strategy, and Phase III tries to prove the optimality of the solution by Phase II. Specifically, Phase I utilizes a modified Hoffman heuristic (MHH) to obtain a high-quality UB. If the UB is equal to the lower bound at root (LB_{root}), this UB is the optimal number of the stations. If the optimal solution is not proved, the cyclic best-first search strategy is executed to attempt to update the UB. Subsequently, breadth-first search strategy is utilized to try to prove the optimality of the solution achieved by Phase II.

Algorithm 1: Main procedure of BBR method

```
% Phase I
Step 1:  Obtain UB using high-performing heuristic;
Step 2:  Achieve the lower bound at the root or $\mathrm{LB}_{root}$;
Step 3:  If $\mathrm{UB} = \mathrm{LB}_{root}$ then
             Terminate this procedure;
         Else execute Step 4;
         Endif
% Phase II
Step 4:  If the termination criterion is met or the optimal solution is found then
             Terminate this procedure;
         Else execute the cyclic best-first search strategy and update UB when necessary;
         Endif
Step 5:  If $\mathrm{UB} = \mathrm{LB}_{root}$ or the termination criterion is met then
             Terminate this procedure;
         Else execute Step 6;
         Endif
```

```
% Phase III
Step 6:  If the termination criterion is satisfied or optimal solution is found then
             Terminate this procedure;
         Else execute breadth-first search strategy and update UB when necessary;
         Endif
```

In the BBR method, all the sub-problems are stored in memory (including the searched sub-problems) in Phase II or Phase III. Firstly, a sub-problem is first selected utilizing the corresponding search strategy. Subsequently, a number of new sub-problems at deeper depth are generated. One sub-problem is abandoned if it cannot achieve smaller upper bound proved by lower bounding methods or it is dominated by other sub-problems proved by the dominance rules. All the remained sub-problems surviving from the lower bounding methods and dominance rules are stored in the memory. Finally, a new sub-problem is again chosen, and this procedure is terminated when $\mathrm{UB} = \mathrm{LB}_{root}$ or the termination criterion is satisfied.

The main search procedure utilized in Step 4 and Step 5 is illustrated in Algorithm 2. For clarity, a sub-problem or a partial solution is donated as $\wp = (A, U, S_1, S_2, \cdots, S_m)$, where A is the set of assigned tasks to the former m stations, U is the set of unallocated tasks, and S_m is the set of tasks allocated to station m. It is clear that $A = \cup_{n=1}^{m} S_j$ and $\mathrm{U} = \mathrm{T} \backslash \mathrm{A}$, where T is the set of all tasks.

Algorithm 2: Main search procedure

```
1   While the termination criterion is not met
2      Select a new non-dominated partial solution $X\,(A, U, S_1, S_2, \cdots, S_m)$ with the search strategy
3      While the number of generated station loads is less than a given number or search tree is not empty
4         Generate a new partial solution $Y\ (A', U', S_1, S_2, \cdots, S_m, S_{m+1})$ in deeper depth utilizing the branching method
5         If $Y$ is a complete solution, update UB when necessary.
6         Delete $Y$ if $\max\{LB1, LB2, LB3\} \geq UB$.
7         Delete $Y$ if it is dominated by one of the dominance rules.
8         Store the sub-problem $Y$.
9         If $\mathrm{BPLB} \geq UB$, sub-problem $Y$ is marked with dominated partial solution.
          % BPLB is the bin-packing lower bound
10     Endwhile
11  Endwhile
```

3.2 Branching

Branching method is proposed to partition of the original problem into a set of smaller sub-problems. This study employs the station-oriented branching method here due to its superiority as demonstrated in published studies (Morrison et al., 2014; Sewell and Jacobson, 2012). The main feature of station-oriented branching method is that it generates a set of complete station loads. Supposed a selected sub-problem is $\wp = (A,U,S_1,S_2,\cdots,S_m)$, station-oriented branching method generates a set of sub-problems $\wp' = (A',U',S_1,S_2,\cdots,S_m,S_{m+1})$ at deep depth. Recall that, for one new sub-problem, the precedence constraint and cycle time constraint must be satisfied, e.g. expressions (4-5) in Section 2.2 must be satisfied for the assigned tasks and $\sum_{i \in S_n} t_i \leq CT \;\; \forall n = 1,2,\cdots,m, m+1$.

3.3 Upper bounds

This research employs the MHH (Li et al., 2018; Morrison et al., 2014; Sewell and Jacobson, 2012). Although MHH is developed by ALBP, this algorithm is adapted for DLBP to obtain a high-quality UB. MHH starts with generating a number of complete stations loads (set to 1000) for station 1, and then the one with the maximum value of $\sum_{i \in S_1} (t_i + \alpha \cdot w_i + \beta \cdot |F_i| - \gamma)$ is selected, and this procedure is terminated until a complete solution is achieved. Here, $F_i (F_i^*)$ is the set of immediate (all) successors of task i, w_i is the positional weight of tasks ($w_i = t_i + \sum_{j \in F_i^*} t_j$), and α, β and γ are three input parameters. Notice that a task j is the successor of task i when task i the AND predecessor or OR predecessor of task j. This expression encourages a full workload with t_i, encourage the tasks with large positional weight or larger number of successors with $\alpha \cdot w_i$ or $\beta \cdot |F_i|$, and also encourage the tasks with larger operation times with $-\gamma$.

Another underlying important problem is determining the values of the three input parameters in order to optimize upper bound. Following the published papers (Li et al., 2018; Morrison et al., 2014; Sewell and Jacobson, 2012), this research sets to values of these parameters as the: $\alpha \in \{0, 0.005, 0.010, 0.015, 0.020\}$, $\beta \in \{0, 0.005, 0.010, 0.015, 0.020\}$ and $\gamma \in \{0,0.01,0.02,0.03\}$. As different combination of the values might lead to different solutions, this research tests all the possible combinations (100 combinations) of the parameter values, and the minimum station number is regarded as the UB by MHH. However, MHH terminates once the station number by a combination is equal to LB_{root} to avoid wasted computation time.

3.4 Lower bounds

Due to the difference in the precedence relations between ALBP and DLBP, the precedence-based lower bounds in ALBP are not applicable to DLBP or need big modifications. Hence, this research mainly employs the bin-packing based lower bounds which are obtained by slacking the DLBP into bin-packing problem. The applied lower bounds include well-known LB1, LB2 and LB3, and the bin-packing lower bound (BPLB) by solving the bin-packing problem optimally with a separate branch and bound solver (Li et al., 2018; Morrison et al., 2014; Sewell and Jacobson, 2012). The first three lower bounds are calculated using expressions (7-10), where $\lceil A \rceil$ is the minimum integer number larger than A, $|\{i \in T | t_i > CT/2\}|$ is the number of tasks satisfying $t_i > CT/2$ under given a set of tasks T = {1, 2,…, N}.

$$\mathrm{LB1} = \left\lceil \textstyle\sum_{i \in T} t_i / CT \right\rceil \tag{7}$$

$$\mathrm{LB2} = |\{i \in T | t_i > CT/2\}| + \left\lceil \frac{|\{i \in T | t_i = CT/2\}|}{2} \right\rceil \tag{8}$$

$$\mathrm{LB3} = \left\lceil \textstyle\sum_{i \in T} v_i \right\rceil \tag{9}$$

$$v_i = \begin{cases} 1 \ \text{ if } \ t_i > 2 \cdot CT/3 \\ 2/3 \ \text{ if } \ t_i = 2 \cdot CT/3 \\ 1/2 \ \text{ if } \ CT/3 < t_i < 2 \cdot CT/3 \\ 1/3 \ \text{ if } \ t_i = CT/3 \end{cases} \tag{10}$$

Regarding BPLB, a separate branch and bound solver is applied to obtain the optimal solutions as the bin-packing problem is already NP-hard. To avoid tremendous time utilized to calculate the BPLB, this branch and bound solver terminates when computation time reaches one second or the total number of loads generated for one bin exceeds 50 (Sewell and Jacobson, 2012). As LB1, LB2 and LB3 are much faster than BPLB, LB1, LB2 and LB3 are first applied and BPLB is later applied to the sub-problems that are not pruned by LB1, LB2 and LB3.

3.5 Dominance rules

This research extends or modifies the four dominance rules in solving ALBP (Li et al., 2018; Morrison et al., 2014; Sewell and Jacobson, 2012) to solve DLBP, including maximum load rule, modified Jackson dominance rules, no-successor rule and memory-based dominance rule. These dominance rules are clarified as follows.

Maximum load rule: A sub-problem $\wp = (A, U, S_1, S_2, \cdots, S_m)$ is pruned when a task i can be assigned to station m while satisfying cycle time constraint and precedence constraint.

Due to OR precedence relation, the modified Jackson dominance (Li et al., 2018; Morrison et al., 2014; Sewell and Jacobson, 2012) cannot be applied to DLBP. Hence, this research develops two modified Jackson dominance rules for two situations. If task i, task j and all the successors of these two tasks are not the OR predecessor of other tasks, the original modified Jackson dominance 1 is applied.

Modified Jackson dominance rule 1**:** A sub-problem $\wp = (A, U, S_1, S_2, \cdots, S_m)$ is pruned when 1) task i, task j and all the successors of these two tasks are not the OR predecessor of other tasks; 2) there is a task i in S_m and an unallocated task j such that $t_i \leq t_j$ and $F_i^* \subseteq F_h^*$; 3) task j can replace task i without violation of cycle time constraint and precedence constraint.

Nevertheless, if task i, task j or one of the successors of these two tasks is the OR predecessor of other tasks, the modified Jackson dominance 2 is applied.

Modified Jackson dominance rule 2: A sub-problem $\wp = (A, U, S_1, S_2, \cdots, S_m)$ is pruned when 1) task i, task j or one of the successors of these two tasks is the OR predecessor of other tasks; 2) there is a task i in S_m and an unallocated task j such that $t_i \leq t_j$ and they have the same successor relation; 3) task j can replace task i without violation of cycle time constraint and precedence constraint.

No-successor rule: A sub-problem $\wp = (A, U, S_1, S_2, \cdots, S_m)$ is pruned when 1) the tasks in S_m have no successors; 2) there is one unassigned task which is assignable has at least one successor.

Memory-based dominance rule: A sub-problem $\wp = (A, U, S_1, S_2, \cdots, S_m)$ is pruned if there is a subproblem $\wp = (A', U', S_1, S_2, \cdots, S_n)$ in memory such that $A = A'$ and $m \geq n$.

As different lower bounding methods and dominance rules need different running time, the sequence of applying them is a vital problem. In this paper, the sequence of applying these dominance rules are presented as follows: for a new sub-problem, LB1, LB2 and LB3 is first applied, later maximum load rule, modified Jackson dominance rules and no-successor rule, subsequently memory-based dominance rule is applied. If this new sub-problem is still not dominated, the BPLB is finally applied due to large computation time.

3.6 Search strategy

This section describes the utilized cyclic best-first search strategy (CBFS) in Phase II and breadth-first search strategy(BrFS) in Phase III. The main procedure of CBFS is presented in Algorithm 3. Firstly, CBFS select the most promising problem at depth 1 and generate a number of children sub-problems for depth 2; the non-dominated children are stored in memory, and later it selects the most promising problem at depth 2 and generates a number of children sub-problems for depth 3. This above procedure terminates when reaching the deepest level, and then it comes back to depth 1 and this cycle is repeated until the termination criterion is met. To select the most promising sub-problems, this research utilizes the selection criterion as $b(\wp) = \mathrm{LB(U)} + {}^{I}/m - \lambda|U|$, where LB(U) is the best lower bound by the lower bounding methods for the unassigned task set U, I is the total idle on the former opened m stations, $|U|$ is the number of unassigned tasks, and λ is an input number set to 0.02. Clearly, the sub-problem with a smaller lower bound, smaller total idle time and larger number of remained tasks (easier to pack) is regarded as the most promising sub-problem.

```
Algorithm 3: Procedure of CBFS
Step 1: Set $l$=1 and generate a number of sub-problems at depth 1;
Step 2: Select the subproblem $\wp$ at depth $l$ with the minimum value of $b(\wp)$;
% $b(\wp)$ is the fitness of a sub-problem $\wp$.
Step 3: Generate a number of children sub-problems at depth $l$+1 and store the non-dominated children;
Step 4: Update $l$ with $l = (l+1) \% (\mathrm{UB}-1)$ and go to Step 2 when there are still unexplored sub-problems; otherwise, terminate the CBFS procedure.
```

To avoid many sub-problems at former depths and obtain high-quality complete solution fast, this research makes a small modification on the CBFS, referred to as MCBFS, following Li et al. (2018). Namely, if the number of subproblems at depth l+1 is larger than 10,000, the sub-problem at depth *l* is not selected and *l* is updated directly utilizing *l*= (*l*+1) % (UB-1). The basic logic behind this modification is that, if there are enough promising subproblems at depth *l*+1, there is no need to generate more sub-problems for depth *l*+1.

Regarding the BrFS, it generates all the sub-problems at depth 1, and later generates all the sub-problems at depth 2, and this procedure is repeated for deeper depths until the optimal solution is achieved. Clearly, BrFS is heavy and might cost a lot of running time to test all the sub-problems at one depth. However, BrFS might be utilized to prove the optimality of the solutions for some instances by Phase II when utilizing the tighter UB achieved by Phase II (Sewell and Jacobson, 2012).

4. Computational study

This section tests the performance of the proposed BBR method on two sets of instances: instances with only AND precedence relation and instances with AND precedence and OR precedence relation. Table 2 presents the tested instances, where the former three instances only have AND precedence relations and latter 11 instances by adding '-OR' have both AND precedence and OR precedence relations. For each instance, the original cycle time and several cycle times within t_{Max} and $\left\lceil \sum_{i \in T} t_i/2 \right\rceil$ are randomly generated (a total number of 20 cycle times) are tested, where t_{Max} is the maximum value of the operation times. Notice that the very small-size instances are not tested as BBR is capable of obtaining the optimal solutions very fast. For P22-OR, P34-OR, P47-OR, P60-OR and P73-OR, the original operation times of several tasks are set to 0 in Kalaycılar et al. (2016), and hence this research re-generates the operation times from a discrete uniform distribution [1,20] following the method in this paper. In addition, this research also generates two large-size instances: P120-OR and P133-OR. P120-OR is generated by combing the precedence relations of P47-OR and P73-OR; P133-OR is generated by combing the precedence relations of P60-OR and P73-OR. As table 2 contains 13 instances and one instance has 20 cycle times for one instance, there are a total number of 260 cases tested here.

Table 2 Summary of the tested instances

Instances	Description
P25	A Samsung SCH-3500 cell phone with 25 subassemblies by Kalayci and Gupta (2013a).
P40	HG5-20 triaxial five speed mechanical transmission with 40 tasks (12 components and 28 fasteners) by Ren et al. (2018c).
P47	A laptop with 47 parts with three set of operation times by Kalayci et al. (2015), referred to as P47A, P47B and P47C.
P10-OR	The instance with 10 tasks by McGovern and Gupta (2003).
P22-OR	The instance modified from ball-point pen with 22 tasks by Kalaycılar et al. (2016).
P34-OR	The instance modified from radio with 34 tasks by Kalaycılar et al. (2016).
P47-OR	The instance with 47 tasks by Kalaycılar et al. (2016).
P60-OR	The instance with 60 tasks by Kalaycılar et al. (2016).
P73-OR	The instance with 73 tasks by Kalaycılar et al. (2016).
P120-OR	Generated by combing the precedence relations of P47-OR and P73-OR.
P133-OR	Generated by combing the precedence relations of P60-OR and P73-OR.

Before solving these instances, the precedence diagram is adjusted so that the task number of the predecessor is less than the task numbers of the successors. Also, dummy tasks are necessary and utilized to transfer the precedence diagram with the complex AND within OR precedence relations, e, g. the dummy task A1 in Fig.1. The BBR is coded utilizing the C++ programing language in Microsoft Visual Studio 2015, and it is compared with the model in Section 2.2 utilizing the CPLEX solver in the General Algebraic Modeling System 23.0. All the experiments are conducted on a personal computer equipped with Intel i7-4790S 3.20 GHz CPU with 8 GB RAM.

4.1 Case studies

This section solves three instances with different precedence relations utilizing the proposed BBR method. Figure 2 illustrates the precedence diagram of Samsung SCH-3500 cell phone (P25) with 25 disassembly tasks, where there are only AND precedence relations. Figure 3 presents the precedence diagram of a ball-point pen with 22 tasks (P22-OR) in Kalaycılar et al. (2016), where there are both AND precedence and OR precedence relations. Figure 4 illustrates the precedence diagram of the instance with 47 tasks (P47-OR) by Kalaycılar et al. (2016), where there are AND precedence, OR precedence and complex OR within AND precedence relations. For instance, there is OR within AND precedence relation before task

10 (see Figure 4).

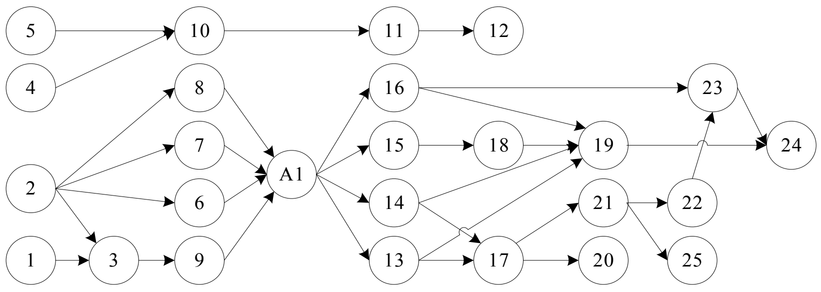

Fig. 2 Precedence diagram of P25 with 25 tasks

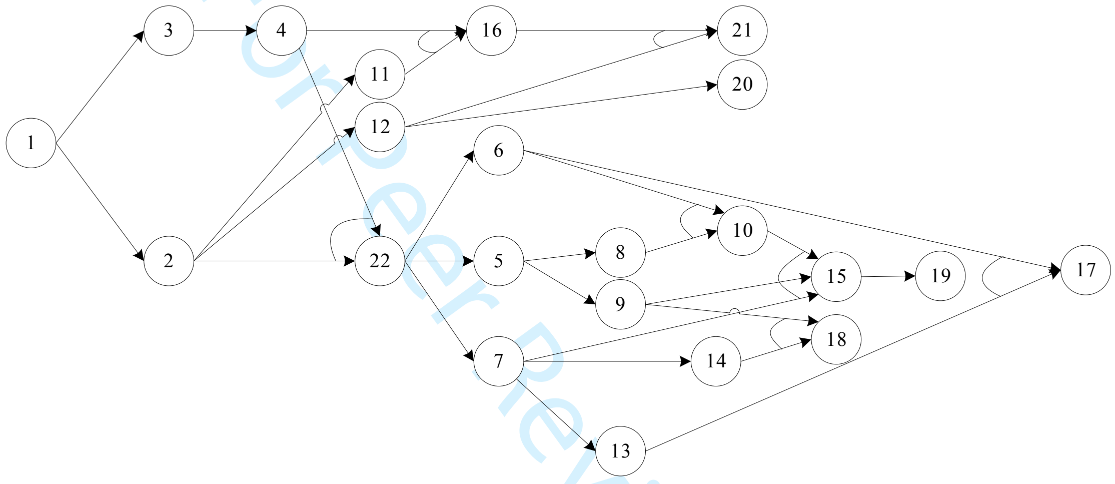

Fig. 3 Precedence diagram of P22-OR with 22 tasks

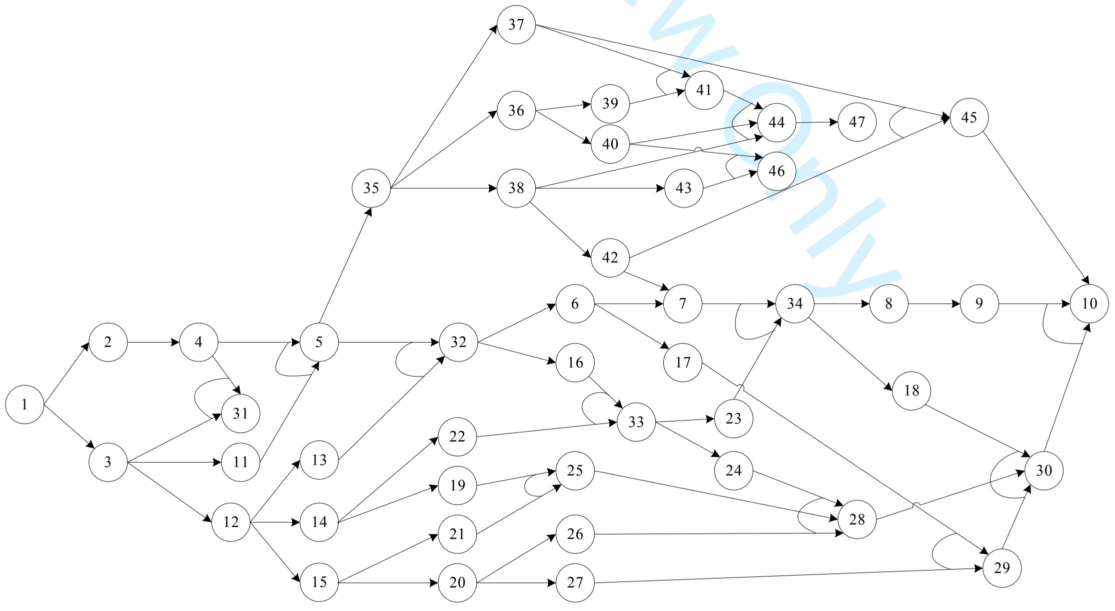

Fig. 4 Precedence diagram of P47-OR with 47 tasks

Table 3 illustrates the optimal solutions (verified by CPLEX solver) for P25 with a cycle time of 18, P22-OR with a cycle time of 39 and optimal solution for P47-OR with a cycle time of 66 by the proposed BBR method. In this table, the second column presents the assigned tasks, and the third and fourth columns show the corresponding operation times and the total operation time of tasks in the second column. After checking the detailed task assignment, it is observed that both the cycle time constraint and precedence constraint are satisfied. It can be concluded that the proposed BBR is capable of solving the DLBPs with OR precedence relations and complex OR within AND precedence relation optimally.

Table 3 Detailed task assignments for the optimal solutions

	P25 with a cycle time of 18			
	Assigned tasks	Operation times	Total time	Optimal station number
Station 1	2,6	2,15	17	9
Station 2	1,7	3,15	18	
Station 3	3,8	3,15	18	
Station 4	9,13	15,2	17	
Station 5	4,14,15,16,17	10,2,2,2,2	18	
Station 6	5,18,20	10,3,5	18	
Station 7	19	18	18	
Station 8	10,11,12,21,22,25	2,2,2,1,5,2	14	
Station 9	23,24	15,2	17	
	P22-OR with a cycle time of 39			
	Assigned tasks	Operation times	Total time	Optimal station number
Station 1	1,3,4	10,12,17	39	7
Station 2	2,22,11,7,15	6,19,1,9,4	39	
Station 3	5,6,10	9,19,11	39	
Station 4	8,9,13,14	15,6,12,6	39	
Station 5	12,18,21	19,6,14	39	
Station 6	16,17,20	17,10,7	34	
Station 7	19	16	16	
	P47-OR with a cycle time of 66			
	Assigned tasks	Operation times	Total time	Optimal station number
Station 1	1,2,3,4,11,12,31,13	5,11,18,4,7,1,4,16	66	8
Station 2	5,14,15,32,35,19	12,6,6,20,7,15	66	
Station 3	6,16,20,21,22,25,36,26	7,7,3,16,11,5,16,1	66	
Station 4	17,27,28,37,39	18,10,17,4,17	66	
Station 5	29,30,33,38,45,24	11,4,20,18,7,6	66	
Station 6	10,23,40,42,44	15,8,9,15,19	66	
Station 7	7,34,41,43,46,47,18	8,19,9,15,5,2,8	66	
Station 8	8,9	18,1	19	

4.2 Comparative study

This section evaluates the performance of BBR method by comparing with CPLEX solver and five recent and effective metaheuristics on a total number of 260 cases. These metaheuristics include: genetic algorithm (GA), partial swarm optimization(PSO), simulated annealing algorithm(SA), tabu search algorithm (TS) and artificial bee colony algorithm (ABC). Additionally, the tested methods are evaluated under six CPU time limits (0.5 second, 1 second, 2 seconds (s), 10s, 100s and 3600s) to have a better observation of the performance of the BBR method under different termination criterion. The detailed procedures of the reimplemented metaheuristics are available upon request. Here, the CPLEX solver terminates when the optimal solution is achieved, or the computation time reaches the given time limit. The metaheuristics terminates when the obtained UB is equal to $\max\{LB1, LB2, LB3\}$ at the root or the computation time reaches the given time limit the as metaheuristics cannot prove the optimality of the achieved solution. This research divides the tested instances in two sets: small-size instances with task number less than or equal to 73 and large-size instances including P120-OR and P133-OR.

Table 4 presents the overall results by the tested methods, #OPT is the number of optimal solutions found, ARPD is the average value of relative percentage deviation (RPD). The RPD is calculated with $100 \cdot (f_{some} - f_{Best})/f_{Best}$, where f_{some} is the number of stations achieved by one method and f_{Best} is the minimum number of stations yielded by all the tested methods. As metaheuristic might produce different solutions in different runs, in this study metaheuristic solve each instance for 20 times, and this table reports the average value in 20 times' independent runs. The detailed results are not exhibited for space limits and they are available upon request.

From this table, it is observed that BBR is the best performer under six CPU time limits. Specifically, BBR is capable of achieving all the optimal solution within 1s whereas the five algorithms and CPLEX can only achieve part of the optimal solutions within 100s. And it is also observed that reimplemented metaheuristics outperform the CPLEX in both solution quality and search speed, especially for large-size instances. Regarding the computation times, BBR only consumes 0.036s on average to find all the optimal solutions and verify the optimality of the achieved solutions. The metaheuristics, on the contrary, cannot verify the optimality of the achieved solutions and cannot find all the optimal solutions even with increased computation time. In summary, this comparative study demonstrates the superiority of the proposed BBR method over the implemented metaheuristic and CPLEX in both solution quality and search speed.

Table 4 The overall results by the tested methods

Time limit (s)	Method	Small-size instances			Large-size instances			All instances		
		#OPT	ARPD	Time(s)	#OPT	ARPD	Time(s)	#OPT	ARPD	Time(s)
0.5	CPLEX	156	-	0.441	0	-	0.500	156	-	0.450
	GA	215.05	0.18	0.025	32.2	0.93	0.102	247.25	0.30	0.037
	PSO	214.75	0.23	0.029	33.4	0.49	0.116	248.15	0.27	0.042
	SA	215.4	0.15	0.025	32.05	1.14	0.108	247.45	0.30	0.037
	TS	215.45	0.15	0.024	32.1	0.92	0.105	247.55	0.27	0.037
	ABC	215.15	0.16	0.025	32.15	1.16	0.102	247.3	0.32	0.037
	BBR	215	0.27	0.035	**39**	**0.04**	**0.026**	**254**	**0.23**	**0.034**
1	CPLEX	170	-	0.687	0	-	1.000	170	-	0.735
	GA	215.35	0.16	0.048	32.1	0.91	0.202	247.45	0.27	0.072
	PSO	214.7	0.24	0.053	33.3	0.75	0.203	248	0.32	0.076
	SA	216.1	0.12	0.045	32	0.86	0.208	248.1	0.24	0.070
	TS	216	0.13	0.046	32.05	1.21	0.206	248.05	0.30	0.070
	ABC	215.7	0.14	0.046	32.05	1.03	0.205	247.75	0.28	0.070
	BBR	**220**	**0.00**	**0.038**	**40**	**0.00**	**0.026**	**260**	**0.00**	**0.036**
2	CPLEX	180	-	1.066	0	-	2.000	180	-	1.210
	GA	215.85	0.14	0.090	32.1	0.98	0.402	247.95	0.27	0.138
	PSO	214.7	0.25	0.103	33.45	0.92	0.363	248.15	0.36	0.143
	SA	216.2	0.12	0.086	32.15	1.10	0.404	248.35	0.27	0.135
	TS	216.25	0.12	0.087	32.1	0.55	0.402	248.35	0.19	0.136
	ABC	215.85	0.13	0.089	32.05	1.20	0.403	247.9	0.30	0.137
	BBR	**220**	**0.00**	**0.038**	**40**	**0.00**	**0.026**	**260**	**0.00**	**0.036**
10	CPLEX	200	-	2.296	0	-	10.000	200	-	3.482
	GA	216.65	0.10	0.403	32.5	0.65	1.910	249.15	0.19	0.635
	PSO	214.6	0.24	0.487	33.55	0.71	1.658	248.15	0.31	0.667
	SA	217.6	0.07	0.376	32.25	0.89	1.981	249.85	0.20	0.623
	TS	217.4	0.08	0.382	32.4	0.99	1.955	249.8	0.22	0.624
	ABC	216.75	0.10	0.396	32.4	1.11	1.947	249.15	0.25	0.634
	BBR	**220**	**0.00**	**0.038**	**40**	**0.00**	**0.026**	**260**	**0.00**	**0.036**
100	CPLEX	216	0.32	7.301	0	-	100.000	216	-	21.562
	GA	218.9	0.03	3.380	33.7	0.42	17.540	252.6	0.09	5.558
	PSO	214.9	0.22	4.710	33.85	0.78	15.442	248.75	0.31	6.361

	SA	219.7	0.01	3.153	33.65	0.63	17.175	253.35	0.11	5.310
	TS	219.4	0.02	3.174	33.5	0.51	17.457	252.9	0.09	5.371
	ABC	218.55	0.04	3.373	33.8	0.69	17.276	252.35	0.14	5.512
	BBR	**220**	**0.00**	**0.038**	**40**	**0.00**	**0.026**	**260**	**0.00**	**0.036**
	CPLEX	218	0.05	91.392	3	-	3508.526	221	-	617.105
	GA	220	0.00	99.181	35.3	0.50	485.073	255.3	0.08	158.549
	PSO	216.1	0.16	155.584	34.5	0.86	522.340	250.6	0.27	212.008
3600	SA	220	0.00	98.659	35.7	0.30	429.442	255.7	0.05	149.549
	TS	220	0.00	98.926	35.75	0.26	430.037	255.75	0.04	149.866
	ABC	220	0.00	98.992	35.85	0.30	423.760	255.85	0.05	148.956
	BBR	**220**	**0.00**	**0.038**	**40**	**0.00**	**0.026**	**260**	**0.00**	**0.036**

*Best in bold

Let us focus on the exact methods, CPLEX and BBR, and table 5 presents the summary of the results by CPLEX solver and BBR method for both small-size and large-size instances under 3600's time limit. In this table, #OPT found is the number of optimal solutions found, #OPT verified is the number of optimal solutions verified, and Max-time and Average-time are the maximum value and the average value of the running times when solving the tested cases. Regarding the small-size instances, it is observed that BBR is capable of obtaining and verifying all the 220 optimal solutions within an average running time of 0.038s. For the most difficult case, BBR only consumes 1.591s. The CPLEX solver, on the contrary, needs much more running time and the average running time by CPLEX solver is 91.392 s. And CPLEX finds 218 optimal solutions among which 215 optimal solutions are verified. This study also conducts the Wilcoxon matched-pairs signed rank test on the consumed running times, and statistical results demonstrate that the proposed BBR is statistically fast than CPLEX solver with a *p* value less than 0.0001. Regarding the large-size instances, it is observed that BBR also achieves the clear superiority over the CPLEX. CPLEX only find three optimal solutions and cannot find optimal solutions for the majority of the large-size instances. The BBR, on the contrary, obtain all the optimal solutions and verify the optimality of all these solutions. In particular, the CPLEX only consumes little computation time and the maximum running time is only 0.948s.

In summary, BBR is capable of solving all the 260 small-size and large-size instances optimally in very short computation time. The CPLEX fails to solve the large-size instances optimally within the given termination criterion and costs much more computation time. All the computational results verify that the proposed BBR outperforms CPLEX solver in solution quality and search speed, and BBR is very fast in solving all the tested instances optimally.

Table 5 Summary of the results by CPLEX solver and BBR method

Instances	Method	#OPT found	#OPT verified	Max-time	Average-time
Small-size instances	CPLEX	218	215	3600	91.392
	BBR	220	220	1.591	0.0380
Large-size instances	CPLEX	3	3	3600	3508.526
	BBR	40	40	0.948	0.0263
All instances	CPLEX	221	218	3600	617.105
	BBR	260	260	1.591	0.0362

5. Conclusions, managerial insights and future researches

This research develops the BBR algorithm an exact methodology to tackle the DLBP with AND/OR precedence to minimize the number of opened stations. The proposed BBR method employs modified Hoffman heuristic to obtain a high-quality upper bound and four bin-packing based lower bound methods to prune the new sub-problems. Furthermore, it utilizes the memory-based dominance rule to eliminate the reduplicated sub-problems by storing all the searched sub-problems and proposes the modified cyclic best-first search strategy with proper station load selection criterion to obtain high-quality complete solutions fast. The proposed BBR method is compared with the mathematical model using CPLEX solver on a set of 260 instances taken from the literature.

Computational results show that BBR obtains optimal solutions for all the tested instances within 0.1 second on average, and the maximum running time during solving these instances is only 1.591 seconds. Moreover, comparative study verifies that the BBR outperforms the CPLEX solver and well-known metaheuristics in both solution quality and search speed aspects. As well as computational advantages, the optimal solutions obtained by the BBR algorithm have provided certain insights for managers; *(i)* it can offer optimal line efficiency to manufacturing engineers in terms of the number of stations for current cycle time; *(ii)* it can prevent to use unnecessary resources such as, workers, machine and tools; *(iii)* it can help to reduce overall cost on the products.

Future revenue stems from developing more dominance rules or lower bounding methods to extending the branch, bound and remember algorithm to tackle the DLBPs with the Successor OR type precedence relations. As there are limited instances, it is quite important to obtain more large-size instances from real industry. It is also suggested to extend the proposed methodology to DLBP with different layouts (U-shaped or two-sided), and the DLBP under uncertainty, e.g. stochastic DLBP.

Acknowledgment

This project is partially supported by National Natural Science Foundation of China under grant 61803287 (Z. L.).

References

Agrawal, S., and Tiwari, M. K. (2008). A collaborative ant colony algorithm to stochastic mixed-model U-shaped disassembly line balancing and sequencing problem. International Journal of Production Research, 46(6), 1405-1429. doi:10.1080/00207540600943985.

Altekin, F. T., and Akkan, C. (2012). Task-failure-driven rebalancing of disassembly lines. International Journal of Production Research, 50(18), 4955-4976. doi:10.1080/00207543.2011.616915.

Altekin, F. T., Kandiller, L., and Ozdemirel, N. E. (2008). Profit-oriented disassembly-line balancing. International Journal of Production Research, 46(10), 2675-2693. doi:10.1080/00207540601137207

Altekin, F., (2017). A comparison of piecewise linear programming formulations for stochastic disassembly line balancing. Int. J. Prod. Res. 55(24), 7412-7434.

Avikal, S., Mishra, P. K., and Jain, R. (2014). A Fuzzy AHP and PROMETHEE method-based heuristic for disassembly line balancing problems. International Journal of Production Research, 52(5), 1306-1317.

Aydemir-Karadag, A., & Turkbey, O. (2013). Multi-objective optimization of stochastic disassembly line balancing with station paralleling. Computers & Industrial Engineering, 65(3), 413-425.

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.

Borba, L., Ritt, M., and Miralles, C. (2018). Exact and heuristic methods for solving the Robotic Assembly Line Balancing Problem. European Journal of Operational Research, 270(1), 146-156. doi:10.1016/j.ejor.2018.03.011

Bentaha, M. L., Battaïa, O., and Dolgui, A. (2014). A sample average approximation method for disassembly line balancing problem under uncertainty. Computers & Operations Research, 51, 111-122.

Bentaha, M. L., Battaïa, O., and Dolgui, A. (2015). An exact solution approach for disassembly line balancing problem under uncertainty of the task processing times. International Journal of Production Research, 53(6), 1807-1818.

Ding, L.-P., Feng, Y.-X., Tan, J.-R., and Gao, Y.-C. (2010). A new multi-objective ant colony algorithm for solving the disassembly line balancing problem. The International Journal of Advanced Manufacturing Technology, 48(5), 761-771. doi:10.1007/s00170-009-2303-5

Güngör, A., and Gupta, S. M. (1999). Disassembly Line Balancing. Paper presented at the Proceedings of the Annual Meeting of the Northeast Decision Sciences Institute, Newport, RI.

Gungor, A., and Gupta, S. M. (2001). A solution approach to the disassembly line balancing problem in the presence of task failures. International Journal of Production Research, 39(7), 1427-1467. doi:10.1080/00207540110052157

Güngör, A., and Gupta, S. M. (2002). Disassembly line in product recovery. International Journal of Production Research, 40(11), 2569-2589. doi:10.1080/00207540210135622

Hezer, S., and Kara, Y. (2015). A network-based shortest route model for parallel disassembly line balancing problem. International Journal of Production Research, 53(6), 1849-1865.

Ilgın, M.A., Akcay, H., Araz, C., (2017). Disassembly line balancing using linear physical programming. International Journal of Production Research, 55(20), 6108-6119.

Kalayci, C. B., and Gupta, S. M. (2013a). Ant colony optimization for sequence-dependent disassembly line balancing problem. Journal of Manufacturing Technology Management, 24(3), 413-427. doi:doi:10.1108/17410381311318909

Kalayci, C. B., and Gupta, S. M. (2013b). Artificial bee colony algorithm for solving sequence-dependent disassembly line balancing problem. Expert Systems with Applications, 40(18), 7231-7241. doi:http://dx.doi.org/10.1016/j.eswa.2013.06.067

Kalayci, C. B., and Gupta, S. M. (2013c). A particle swarm optimization algorithm with neighborhood-based mutation for sequence-dependent disassembly line balancing problem. The International Journal of Advanced Manufacturing Technology, 69(1), 197-209. doi:10.1007/s00170-013-4990-1

Kalayci, C. B., and Gupta, S. M. (2014). A tabu search algorithm for balancing a sequence-dependent disassembly line. Production Planning and Control, 25(2), 149-160.

doi:10.1080/09537287.2013.782949

Kalayci, C. B., Hancilar, A., Gungor, A., and Gupta, S. M. (2015). Multi-objective fuzzy disassembly line balancing using a hybrid discrete artificial bee colony algorithm. Journal of Manufacturing Systems, 37, 672-682. doi:http://dx.doi.org/10.1016/j.jmsy.2014.11.015

Kalayci, C. B., Polat, O., and Gupta, S. M. (2016). A hybrid genetic algorithm for sequence-dependent disassembly line balancing problem. Annals of Operations Research, 242(2), 321-354. doi:10.1007/s10479-014-1641-3

Kalaycılar, E. G., Azizoğlu, M., and Yeralan, S. (2016). A disassembly line balancing problem with fixed number of workstations. European Journal of Operational Research, 249(2), 592-604. doi:http://dx.doi.org/10.1016/j.ejor.2015.09.004

Koc, A., Sabuncuoglu, I., and Erel, E. (2009). Two exact formulations for disassembly line balancing problems with task precedence diagram construction using an AND/OR graph. IIE Transactions, 41(10), 866-881. doi:10.1080/07408170802510390.

Li, Y. (2017). The type-II assembly line rebalancing problem considering stochastic task learning. International Journal of Production Research, 55(24), 7334-7355.

Li, Y., and Boucher, T. O. (2017). Assembly line balancing problem with task learning and dynamic task reassignment. The International Journal of Advanced Manufacturing Technology, 88(9-12), 3089-3097.

Li, Z., Kucukkoc, I., and Zhang, Z. (2018). Branch, bound and remember algorithm for U-shaped assembly line balancing problem. Computers & Industrial Engineering, 124, 24-35. doi:10.1016/j.cie.2018.06.037

Li, J., Chen, X., Zhu, Z., Yang, C., & Chu, C. (2019). A branch, bound, and remember algorithm for the simple disassembly line balancing problem. Computers & Operations Research.

Liu, J., and Wang, S. (2017). Balancing Disassembly Line in Product Recovery to Promote the Coordinated Development of Economy and Environment. Sustainability, 9(2), 309.

Liu, J., Zhou, Z., Pham, D. T., Xu, W., Yan, J., Liu, A., Liu, Q. (2018). An improved multi-objective discrete bees algorithm for robotic disassembly line balancing problem in remanufacturing. International Journal of Advanced Manufacturing Technology, 97(9-12), 3937-3962. doi:10.1007/s00170-018-2183-7

McGovern, S. M., and Gupta, S. M. (2003). 2-opt heuristic for the disassembly line balancing problem. Paper presented at the Proceedings of the SPIE International Conference on Environmentally Conscious Manufacturing III, Providence, RI.

McGovern, S. M., and Gupta, S. M. (2007). A balancing method and genetic algorithm for disassembly line balancing. European Journal of Operational Research, 179(3), 692-708.

doi:http://dx.doi.org/10.1016/j.ejor.2005.03.055

McGovern, S. M., and Gupta, S. M. (2007). Combinatorial optimization analysis of the unary NP-complete disassembly line balancing problem. International Journal of Production Research, 45(18-19), 4485-4511. doi:10.1080/00207540701476281

Mete, S., Çil, Z. A., Ağpak, K., Özceylan, E., and Dolgui, A. (2016a). A solution approach based on beam search algorithm for disassembly line balancing problem. Journal of Manufacturing Systems, 41, 188-200. doi:http://dx.doi.org/10.1016/j.jmsy.2016.09.002

Mete, S., Çil, Z. A., Özceylan, E., and Ağpak, K. (2016b). Resource constrained disassembly line balancing problem. IFAC-PapersOnLine, 49(12), 921-925.

Mete, S., Çil, Z. A., Celik, E., and Ozceylan, E. (2019). Supply-driven rebalancing of disassembly lines: A novel mathematical model approach. Journal of Cleaner Production, 213, 1157-1164.

Mete, S., Çil, Z. A., Özceylan, E., Ağpak, K., and Battaïa, O. (2018). An optimisation support for the design of hybrid production lines including assembly and disassembly tasks. International Journal of Production Research, 1-15. doi:10.1080/00207543.2018.1428774

Morrison, D. R., Sewell, E. C., and Jacobson, S. H. (2014). An application of the branch, bound, and remember algorithm to a new simple assembly line balancing dataset. European Journal of Operational Research, 236(2), 403-409.

Özceylan, E., Kalayci, C. B., Güngör, A., and Gupta, S. M. (2018). Disassembly line balancing problem: a review of the state of the art and future directions. International Journal of Production Research, 1-23. doi:10.1080/00207543.2018.1428775

Paksoy, T., Güngör, A., Özceylan, E., and Hancilar, A. (2013). Mixed model disassembly line balancing problem with fuzzy goals. International Journal of Production Research, 51(20), 6082-6096. doi:10.1080/00207543.2013.795251

Ren, Y., Yu, D., Zhang, C., Tian, G., Meng, L., and Zhou, X. (2017). An improved gravitational search algorithm for profit-oriented partial disassembly line balancing problem. International Journal of Production Research, 55(24), 7302-7316. doi:10.1080/00207543.2017.1341066

Ren, Y., Zhang, C., Zhao, F., Tian, G., Lin, W., Meng, L., and Li, H. (2018b). Disassembly line balancing problem using interdependent weights-based multi-criteria decision making and 2-Optimal algorithm. Journal of Cleaner Production, 174, 1475-1486. doi:https://doi.org/10.1016/j.jclepro.2017.10.308

Ren, Y., Zhang, C., Zhao, F., Triebe, M. J., and Meng, L. (2018a). An MCDM-Based Multiobjective General Variable Neighborhood Search Approach for Disassembly Line Balancing Problem. IEEE Transactions on Systems, Man, and Cybernetics: Systems, 1-14. doi:10.1109/TSMC.2018.2862827

Ren, Y., Zhang, C., Zhao, F., Xiao, H., and Tian, G. (2018c). An asynchronous parallel disassembly planning based on genetic algorithm. European Journal of Operational Research, 269(2), 647-660. doi:https://doi.org/10.1016/j.ejor.2018.01.055

Sewell, E. C., and Jacobson, S. H. (2012). A Branch, Bound, and Remember Algorithm for the Simple Assembly Line Balancing Problem. INFORMS Journal on Computing, 24(3), 433-442. doi:10.1287/ijoc.1110.0462

Vilà, M., and Pereira, J. (2014). A branch-and-bound algorithm for assembly line worker assignment and balancing problems. Computers & Operations Research, 44, 105-114. doi:http://dx.doi.org/10.1016/j.cor.2013.10.016

Xiao, S., Wang, Y., Yu, H., and Nie, S. (2017). An Entropy-Based Adaptive Hybrid Particle Swarm Optimization for Disassembly Line Balancing Problems. Entropy, 19(11), 596.

Zhang, Z., Wang, K., Zhu, L., and Wang, Y. (2017). A Pareto improved artificial fish swarm algorithm for solving a multi-objective fuzzy disassembly line balancing problem. Expert Systems with Applications, 86, 165-176. doi:https://doi.org/10.1016/j.eswa.2017.05.053

Zhu, L., Zhang, Z., and Wang, Y. (2018). A Pareto firefly algorithm for multi-objective disassembly line balancing problems with hazard evaluation. International Journal of Production Research, 1-21. doi:10.1080/00207543.2018.1471238