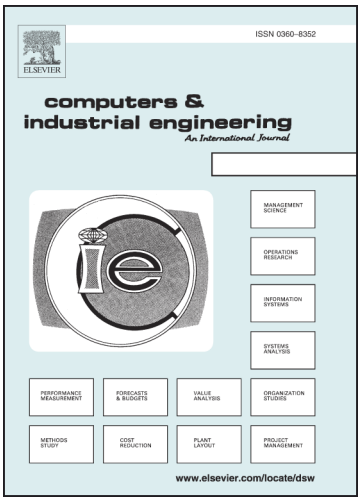
Accepted Manuscript



Branch, bound and remember algorithm for U-shaped assembly line balancing problem

Zixiang Li, Ibrahim Kucukkoc, Zikai Zhang

PII: S0360-8352(18)30315-2
DOI: https://doi.org/10.1016/j.cie.2018.06.037
Reference: CAIE 5300

To appear in: *Computers & Industrial Engineering*

Received Date: 14 January 2018
Revised Date: 26 May 2018
Accepted Date: 29 June 2018

Please cite this article as: Li, Z., Kucukkoc, I., Zhang, Z., Branch, bound and remember algorithm for U-shaped assembly line balancing problem, *Computers & Industrial Engineering* (2018), doi: https://doi.org/10.1016/j.cie.2018.06.037

Branch, bound and remember algorithm for U-shaped assembly line balancing problem

Zixiang Li[1,2], Ibrahim Kucukkoc[3*], Zikai Zhang[1]

[1] Key Laboratory of Metallurgical Equipment and Control Technology, Wuhan University of Science and Technology, China.

[2] Hubei Key Laboratory of Mechanical Transmission and Manufacturing Engineering, Wuhan University of Science and Technology, China.

[3] Industrial Engineering Department, Balikesir University, Cagis Campus, Balikesir 10145, Turkey.

Email: zixiangliwust@gmail.com (Z-X Li); ikucukkoc@balikesir.edu.tr (I. Kucukkoc); zhangzikai0703@gmail.com (Z-K Zhang);

Abstract: This research develops a branch, bound and remember algorithm to address U-shaped assembly line balancing problem. This method proposes a cyclic best-first search strategy which uses memory to preserve searched sub-problems and eliminate redundancy. As the search space in U-shaped assembly lines is much larger than that in simple assembly lines, the search process is easily terminated due to out of memory issue when solving large-size problems. The proposed method employs several improvements, including two new dominance rules, renumbering the tasks when generating the station loads, a new criterion to select the most promising sub-problem and limiting the number of sub-problems at each depth. The proposed methodology is tested on Scholl's well-known 269 benchmark problems and a new data set published in 2013, where backtracking rule is also applied to save memory. Computational comparative study demonstrates that the proposed method outperforms the two current best exact methods (ULINO and branch, price and remember algorithm) by achieving 259 optimal solutions for Scholl's well-known 269 benchmark problems. The proposed method also outperforms the current best branch, price and remember algorithm by optimally solving over 97% of the problems in the new data set.

Keywords: Combinatorial optimization; assembly line balancing; U-shaped assembly line; branch and bound, exact solution

*Corresponding author

Acknowledgment

The second author (I.K.) acknowledges the partial financial support received from the Balikesir University, Scientific Research Projects Department under grant number BAP-2017-179.

Branch, bound and remember algorithm for U-shaped assembly line balancing problem

Abstract: This research develops a branch, bound and remember algorithm to address U-shaped assembly line balancing problem. This method proposes a cyclic best-first search strategy which uses memory to preserve searched sub-problems and eliminate redundancy. As the search space in U-shaped assembly lines is much larger than that in simple assembly lines, the search process is easily terminated due to out of memory issue when solving large-size problems. The proposed method employs several improvements, including two new dominance rules, renumbering the tasks when generating the station loads, a new criterion to select the most promising sub-problem and limiting the number of sub-problems at each depth. The proposed methodology is tested on Scholl's well-known 269 benchmark problems and a new data set published in 2013, where backtracking rule is also applied to save memory. Computational comparative study demonstrates that the proposed method outperforms the two current best exact methods (ULINO and branch, price and remember algorithm) by achieving 259 optimal solutions for Scholl's well-known 269 benchmark problems. The proposed method also outperforms the current best branch, price and remember algorithm by optimally solving over 97% of the problems in the new data set.

Keywords: Combinatorial optimization; assembly line balancing; U-shaped assembly line; branch and bound, exact solution

Nomenclature

SALBP	: The simple assembly line balancing problem
SALBP-1	: Type-1 SALBP (the aim is to minimize the number of workstations given the cycle time)
UALBP	: The U-shaped assembly line balancing problem
UALBP-1	: Type-1 UALBP (the aim is to minimize the number of workstations given the cycle time)
ULINO	: The branch and bound procedure proposed by Scholl and Klein [16] to solve UALBP-1
EUREKA	: The branch and bound procedure proposed by Hoffmann [5] to solve SALBP-1
FABLE	: The branch and bound procedure proposed by Johnson [6] to solve SALBP-1
OptPack	: The branch and bound procedure proposed by Nourie and Venta [7] to solve SALBP-1
SALOME	: The branch and bound procedure proposed by Scholl and Klein [8, 9] to solve SALBP-1
BBR	: The branch, bound and remember algorithm
BPR	: The branch, price and remember algorithm
MHH	: Modified Hoffman heuristic
CBFS	: Cyclic best-first search strategy
BPLB	: Bin packing lower bound
DFS	: Depth-first search strategy
BFS	: Best-first search strategy
BrFS	: Breadth-first search strategy
BBR_BSD	: The BBR method utilizing BFS and dominance rule in Scholl and Klein [16] without renumbering
BBR_BND	: The BBR method utilizing BFS and new dominance rule without renumbering
BBR_CSD	: The BBR method utilizing CBFS and dominance rule in Scholl and Klein [16] without renumbering
BBR_CND	: The BBR method utilizing CBFS and new dominance rule without renumbering
BBR_BNP	: The BBR method utilizing BFS and new dominance rule with renumbering tasks based on positional weight and operation time
BBR_BNO	: The BBR method utilizing BFS and new dominance rule with renumbering tasks based on operation time

BBR_CNP : The BBR method utilizing CBFS and new dominance rule with renumbering tasks based on positional weight and operation time

BBR_CNO : The BBR method utilizing CBFS and new dominance rule with renumbering tasks based on operation time

RPD : Relative percentage deviation

MEJR : The modified extended Jackson rule

NSPR : The no-successors and no-predecessors rule

1. Introduction

Assembly line balancing is an extensively studied optimization problem, which has great implications in modern industry [1, 2]. This problem has many variants as surveyed by Battaïa and Dolgui [2], and the basic edition of this problem is the simple assembly line balancing problem (SALBP). SALBP can be, without loss of generality, defined as the problem of assigning a set of Nt tasks ($T = \{1,2,\cdots,i,\cdots,Nt\}$) to a set of stations with the objective of optimizing one or more performance criterion (i.e. minimizing the cycle time and/or the number of stations). Each task (task i) has a pre-determined, fixed and non-negative operation time (t_i), and the capacity constraint (caused by cycle time) and precedence relationship constraint must be satisfied. Specifically, the total operation time of assigned tasks on each station should be less than or equal to a given time, referred to as cycle time (CT). A task can be assigned upon all of its predecessors (if any) have been allocated to the former station or the former position in the same station. In other words, the immediate (all) predecessors of task i, denoted as $P_i(P_i^*)$, must be allocated to the former station or the former position in the same station.

As a variant of SALBP, U-shaped assembly line balancing problem (UALBP) draws increasing attention from both researchers and engineers due to higher flexibility and productivity [3]. Regarding UALBP with the objective of optimizing the station number, a set of Nt tasks is divided and assigned to a set of stations comprised of the entrance side and exit side. The main feature that differentiates UALBP from SALBP is that a task is assignable when all its predecessors or successors have been allocated.

There are tremendous methods to address the assembly line balancing problems, which can be divided into three categories [2, 4]: exact methods, heuristic methods and meta-heuristic methods. Regarding the exact methods for SALBP, there are many published methods: EUREKA [5], FABLE [6], OptPack [7], SALOME [8, 9], dynamic-programming heuristic [10], and branch, bound and remember (BBR) algorithm [11, 12], to cite just a few. Among them, BBR algorithm is the best performer which finds all the optimal solutions of Scholl's well-known 269 benchmark problems. Regarding the heuristic method, the Hoffmann heuristic or its variants based on the incomplete enumeration are the best performers [11, 13, 14]. Meta-heuristics comprise the majority of the published methods [2], and hybrid beam search and ant colony algorithm are the best performers [15].

As for the applied methods to solve UALBP, there are two exact methods: ULINO [16] and branch, price and remember (BPR) algorithm [17]. BPR is the best exact method, and this method obtains 255 optimal solutions of Scholl's 269 benchmark instances. Regarding the meta-heuristic method, there are many methods to optimize the number of stations, e.g. simulated annealing algorithm [18], ant colony optimization [19, 20], station-oriented ant colony optimization [21]. The recently published station-oriented ant colony optimization [21] is the best metaheuristic, which produces 255 optimal solutions of Scholl's benchmark instances and outperforms ULINO in 21 cases.

As the research on UALBP using exact methods is limited and BBR method is quite effective for

SALBP, this research extends the BBR method to UALBP and improves the BBR method to produce better results, which is the main motivation of this paper. Specifically, this research presents three major contributions as follows. Firstly, BBR method is extended and improved to solve UALBP. This method utilizes modified Hoffman heuristic to achieve high-quality upper bound and cyclic best-first search strategy (CBFS) to find the optimal solution. This research also provides several improvements to enhance the performance of UALBP: two new dominance rules, renumbering the tasks when generating the station loads, new criterion to select the most promising sub-problem and limiting the number of sub-problems at each depth. Secondly, a comprehensive study is carried out where eight BBR methods with different configurations are evaluated on Scholl's 269 benchmark instances and the new 6825 instances in 2013 by Otto and Otto [22]. These BBR methods utilize different dominance rules and search strategies to test the performance of these improvements introduced. Thirdly, tested BBR methods outperform the current ULINO and BPR by achieving many new optimal solutions or upper bounds. For instance, the optimal solution of Scholl's instance with 297 tasks and a cycle time of 1422 (which has been open for almost 20 years) is achieved for the first time. The proposed method also finds 41 new optimal solutions when solving the new data set.

The remainder of this paper is organized as follows. Section 2 provides a description of the considered problem, and Section 3 presents the details of the proposed BBR algorithm. The computational study is carried out in Section 4 and the performance of several BBR methods and two published exact methods are compared. Finally, Section 5 concludes the research and provides several future research directions.

2. Problem description

UALBP with station number minimization criterion can be described as follows: a set of Nt tasks is portioned into the entrance side and exit side of the minimum stations while satisfying the cycle time and precedence relationship constraints. To satisfy the cycle time constraint, the total operation time of tasks in the entrance side and exit side of a station must be less than or equal to the cycle time. With regard to the precedence relationship constraint, a task is assignable when all its predecessors or successors have been assigned. Specifically, a task can be allocated to the entrance side when all its predecessors have been allocated. On the other hand, a task can be allocated to the exit side when all its successors have been allocated.

This section illustrates an example in Fig.1, where precedence relationships among 11 tasks are presented. In this figure, the numbers in the circles denote the tasks and the numbers over the circles indicate the operation times of the tasks (in time units). The tasks in the entrance side and exit side have reverse precedence relationships, where the precedence relationship in the entrance side is the same to that in SALBP.

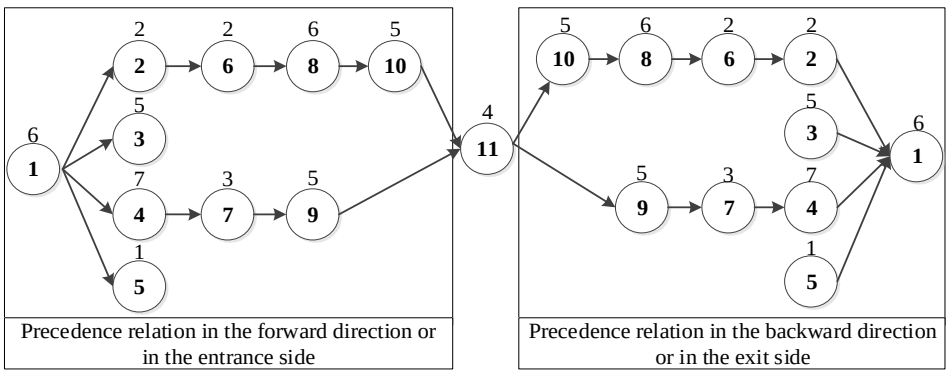


Fig.1 The precedence relationships diagram for a U-shaped assembly line balancing problem

Figure 2 illustrates the detailed task assignment of an example solution with a cycle time fixed to 10

(time units). It is clear that a station is divided into two portions: entrance side and exit side. When there are tasks allocated to both entrance side and exit side, a worker first operates the tasks on the entrance side and moves to the exit side and operate the assigned tasks. This is repeated between the entrance and exit sides in each cycle. The predecessors of a task in the entrance side must be allocated to the former station or the former position of the same station, e.g. task 1 which is the predecessor of task 3 is allocated to station 1 before the station in which task 3 is allocated. On the contrary, the predecessors of a task in the exit side must be allocated to the latter station or the latter position of the same station, e.g. task 11 is allocated to station 1 and its predecessor task 9 is allocated to station 2. It is also observed that, if a pair of tasks with precedence relationship in between is allocated to the entrance side and exit side respectively, the predecessor is allocated to the entrance side and the succeeding task is allocated to the exit side.

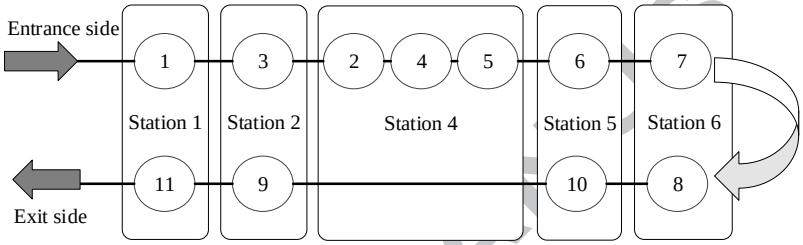


Fig.2 Detailed task assignment for the illustrated example

3. The proposed BBR algorithm

BBR algorithm is an exact method which inherits the main features of branch and bound and dynamic programming [11]. This method has produced state-of-the-art results for SALBP. BBR method stores and remembers all the sub-problems generated during the search process, and then utilizes the memory-based dominance rule to eliminate the duplication of sub-problems.

Due to its superiority, this research extends the BBR method to solve the UALBP. Nevertheless, the search space for UALBP is much larger than that for SALBP and there are more sub-problems in each depth. Initial experiments demonstrate that the simple adaption of BBR might terminate due to out of memory issue when solving large-size instances. Hence, this research also develops several improvements: new pruning rules and dominance rules, renumbering the tasks in generating the station loads, new criterion to select the most promising sub-problem and limiting the number of sub-problems at each depth. The main procedure and segments of BBR are explained as follows.

3.1 The procedure of the proposed method

The procedure of the proposed BBR method is demonstrated as follows, where *LB1*, *LB2*, *LB3* and *BPLB* are the lower bounds (to be explained in Section 3.4). As illustrated in Algorithm 1, this procedure consists of three phases following Sewell and Jacobson [11] and Morrison, Sewell [12].

In Phase I, the BBR method utilizes the modified Hoffman heuristic (MHH), proposed by Sewell and Jacobson [11], to achieve a high-quality upper bound (UB). Recall that MHH is an improved edition of the original Hoffman heuristic proposed by Hoffman [14]. MHH is capable of achieving much better upper bound and hence it is selected to achieve high-quality upper bounds. If UB is equal to the global lower bound at the root (LB_{root}), the optimal solution is obtained and BBR method terminates. Otherwise, BBR executes Phase II to attempt to find the optimal solution using CBFS. In this phase, BBR stores all the sub-problems searched so far and eliminates the dominated station loads using pruning rules and dominance rules. If Phase II cannot prove the optimality of the achieved solution

within given computational time limit, Phase III is employed to attempt to prove the optimality of the found solution using breadth-first search (BrFS). Recall that Phase III does not take effect in our initial experiments, but this research preserves this phase in the case of some special circumstances.

Algorithm 1: BBR algorithm

% Phase I

Step 1: Achieve UB by MHH.

% End of Phase I

Step 2: Obtain global lower bound at the root using $\mathrm{LB}_{root} = max(LB1, LB2, LB3, BPLB)$.

Step 3: If $\mathrm{UB} = \mathrm{LB}_{root}$, terminate; otherwise, go to Step 4.

% Phase II

Step 4: Run CBFS and update UB when smaller UB is achieved. If the termination criterion is satisfied, terminate.

% End of Phase II

Step 5: If $\mathrm{UB} = \mathrm{LB}_{root}$ or termination criterion is satisfied, terminate; otherwise, go to Step 6.

% Phase III

Step 6: Run BrFS and update UB when smaller UB is achieved. If $\mathrm{UB} = \mathrm{LB}_{root}$ or termination criterion is satisfied, terminate this procedure.

% End of Phase III

3.2 Branching

This section illustrates the implemented branching which partitions the original problem into smaller sub-problems. Each partial solution is referred to as $\wp = (A, U, S_1, S_2, \cdots, S_m)$, where A is the assigned tasks to former m stations, U denotes the unassigned tasks ($\mathrm{U} = \mathrm{T}\backslash\mathrm{A}$) and S_m is the set of tasks allocated to station m ($A = \cup_{j=1}^{m} S_j$). A partial solution must satisfy the precedence constraint and cycle time constraint. Specifically, the predecessor h of task i must be allocated ahead of task i in the entrance side ($h \in \cup_{j=1}^{k} S_j$ when $i \in S_k$ in the entrance side) and task i must be allocated ahead of the predecessor h in the exit side ($i \in \cup_{j=1}^{k} S_j$ when $h \in S_k$ in the exit side). Regarding the cycle time constraint, the total operation time on every station should not be larger than a given cycle time ($\sum_{i \in S_j} t_i \leq CT \ \ \forall j = 1,2, \ldots, m$).

There are two popular branching methods: task-oriented branching [6, 7] and station-oriented branching [5, 8, 9, 11, 12]. Task-oriented branching creates sub-problems by allocating a task to the current station or to the next station when it cannot be assigned to the current station. Station-oriented branching obtains a complete load and allocates it to the first available station. Both branching methods have been applied to SALBP, but only station-oriented branching is applied to UALBP [16, 17]. Following Scholl and Klein [16] and Sewell and Jacobson [11] and many others, this research employs the station-oriented branching due to its superiority as proved in published papers. Inspired by the task renumbering in Scholl and Klein [16], this research also renumbers the tasks to obtain the most promising load as early as possible. The tasks are renumbered in the enumeration as that the tasks with both larger positional weight and larger operation time are moved to the former position in the assignable task set when more than one task can be allocated to the entrance side. If more than one task can be allocated to the exit side, the tasks with both larger reverse positional weight and larger operation time are moved to the former position in the assignable task set. This aforementioned method is referred to as renumbering the tasks using positional weight and operation time in Table 1 in Section 4. The method in which tasks with larger operation time are moved to the former position is referred to

as renumbering the tasks using operation time. The proposed BBR method utilizes the first approach to renumber the tasks, but the second approach (based on task operation time) will also be tested in Section 4.

3.3 Upper bounds

For BBR methods, upper bound is an important factor and a high-quality upper bound can increase the search speed and reduce the running time. This research extends the MHH proposed by Sewell and Jacobson [11] to UALBP. The MHH procedure is illustrated in Algorithm 2.

Algorithm 2: MHH procedure

Step 1: If all tasks have been assigned, terminate. Otherwise, go to Step 2.

Step 2: Open a new station.

Step 3: Generate a set of possible station loads (set to 1,000).

Step 4: Select the station load with the maximum value of $\sum_{i\in FS}(t_i + \alpha \cdot w_i + \beta \cdot |F_i| - \gamma) + \sum_{i\in BS}(t_i + \alpha \cdot ow_i + \beta \cdot |P_i| - \gamma)$.

Step 5: Assign this station load to the current open station and go to Step 1.

In this algorithm, FS (BS) is the set of tasks allocated to the current new station in the forward direction (backward) direction. $F_i(F_i^*)$ is the set of immediate (all) successors of task i and $P_i(P_i^*)$ is the set of immediate (all) predecessors of task i. The term w_i denotes the positional weights of tasks in the forward (backward) direction ($w_i = t_i + \sum_{h\in F_i^*} t_h$) and ow_i is the reverse positional weights of tasks in the backward direction ($ow_i = t_i + \sum_{h\in P_i^*} t_h$). The rational of the formula in Step 4 is clarified as follows: the term t_i preserves large workload and encourages a full workload, $\alpha \cdot w_i$ ($\alpha \cdot ow_i$) preserves the tasks whose successors (predecessors) have a larger total operation time, $\beta \cdot |F_i|$ ($\beta \cdot |P_i|$) encourages tasks which make more tasks assignable and $-\gamma$ encourages the tasks with larger operation times.

There are three parameters, α, β and γ, to be determined. Based on the work by Sewell and Jacobson [11] and Morrison, Sewell [12] this research sets the values of these parameters as follows: $\alpha, \beta \in \{0, 0.005, 0.010, 0.015, 0.020\}$, and $\gamma \in \{0, 0.01, 0.02, 0.03\}$. The proposed MHH runs 100 times utilizing each combination of these factors and the minimum station number among them is regarded as the upper bound by MHH.

3.4 Lower bounds

The BBR method utilizes three well-known lower bounds (LB1, LB2 and LB3) [8] and one bin-packing lower bound (BPLB) [11, 12] on SALBP. LB1 and LB2 are defined in Equation (1) and Equation (2), respectively, and LB3 is defined in Equations (3)-(4). Detailed description of these lower bound rules refers to Scholl and Klein [8] and Scholl and Klein [16].

$$\mathrm{LB1} = \lceil \textstyle\sum_{i\in T} t_i / CT \rceil \quad (1)$$

$$\mathrm{LB2} = |\{i \in T : t_i > CT/2\}| + \left\lceil \frac{|\{i \in T : t_i = CT/2\}|}{2} \right\rceil \quad (2)$$

$$\mathrm{LB3} = \lceil \textstyle\sum_{i\in T} v_i \rceil \quad (3)$$

$$v_i = \begin{cases} 1 \text{ if } t_i > 2 \cdot CT/3 \\ 2/3 \text{ if } t_i = 2 \cdot CT/3 \\ 1/2 \text{ if } CT/3 < t_i < 2 \cdot CT/3 \\ 1/3 \text{ if } t_i = CT/3 \end{cases} \quad (4)$$

BPLB is a new lower bound found by slacking the UALBP into the bin packing problem. As the bin packing problem is already NP-hard, this research utilizes a separate branch and bound solver to achieve the optimal solution for the bin packing problem as presented by Sewell and Jacobson [11]. To avoid tremendous time to obtain BPLB, this separate branch and bound solver terminates when an optimal solution is achieved or computational time exceeds one second. This BPLB is quite effective in proving the optimality for many cases, and detailed description of BPLB is first presented by Sewell and Jacobson [11]. When applying these four lower bounds, LB1, LB2 and LB3 are first computed. If the maximum of them is greater than or equal to the incumbent UB, this sub-problem is pruned. Otherwise, BPLB is computed and the sub-problem is pruned when BPLB is greater than or equal to the incumbent UB.

3.5 Dominance rules

Dominance rules are applied to prune the dominated sub-problem. Scholl and Klein [16] employ three dominance rules: maximum load rule, modified Jackson dominance rule and tree dominance rule as follows.

***Maximum load rule*:** A partial solution is pruned if this partial solution contains a station load S_j and a task *i* can be allocated to station *j* without the violation of cycle time constraint and precedence relationship constraint.

Modified Jackson dominance rule: For a given partial solution, a sub-problem containing this partial solution is pruned if 1) there is a task *i* assigned to the last station and an unallocated task *h* such that task *i* is potentially forward dominated by task *h* ($t_i \leq t_h$ and $F_i^* \subseteq F_h^*$), and 2) task *h* can replace task *i* without violation of cycle time constraint and precedence relationship constraint. For a given partial solution, a sub-problem containing this partial solution is pruned if 1) there is a task *i* assigned to the last station and an unallocated task *h* such that task *i* is potentially backward dominated by task *h* ($t_i \leq t_h$ and $P_i^* \subseteq P_h^*$), and 2) task *h* can replace task *i* without the violation of cycle time and precedence relationship constraints. Recall that, task *i* and task *h* do not dominate each other if they have identical operation times and the same set of predecessors in the forward direction or successors in the backward direction.

Tree dominance rule: A partial solution is equivalent to another one if they contain the same set of tasks with different sequence in task assignment. This rule stores the minimum station numbers and corresponding subsets of tasks assigned to stations in partial solutions searched so far. The sub-problem containing this partial solution is pruned if this partial solution is equivalent to a subset of tasks stored and requires no smaller station number.

Yolmeh and Salehi [17] replaced the tree dominance rule with a memory-based dominance rule in implementing BPR algorithm. In this rule, all the sub-problems need to be stored using a hash table. This rule is clarified as follows.

Memory-based dominance rule: The current sub-problem is pruned if the assigned task set of the current sub-problem is equivalent to the task set of a previously identified sub-problem, and the current sub-problem requires no smaller station number.

Both Scholl and Klein [16] and Yolmeh and Salehi [17] utilize the modified Jackson dominance rule,

whereas there might be one possible drawback. Suppose that task *i* in a partial solution is allocated to the entrance side and task *i* is potentially backward dominated by unassigned task *h*. This modified Jackson dominance will prune the original sub-problem even though task *i* is not potentially forward dominated by unassigned task *h*. This situation might lead to achieving wrong optimal solutions, and thus this modified Jackson dominance also adds that task *i* and task *h* do not dominate each other if they have identical operation times and the same set of predecessors in the forward direction or successors in the backward direction. The added limit somewhat repairs the possible drawback. In fact, the preliminary experiments demonstrate that the modified Jackson dominance rule without the added limit produces wrong optimal solutions with larger station number than that of the true optimal solution. However, the modified Jackson dominance rule with the added limit achieves or proves no wrong optimal solutions.

Different from the aforementioned modified Jackson dominance, the proposed BBR employs two new dominance rules modified from those in SALBP [11] as well as maximum load rule and memory-based dominance rule.

Modified extended Jackson rule: For a given partial solution, a sub-problem containing this partial solution is pruned if 1) there is a task *i* assigned to the entrance side of the last station and an unallocated task *h* such that task *i* and task *h* have no precedence relationship, $t_i \leq t_h$ and $F_i^* \subseteq F_h^*$; and 2) task *h* can be assigned to the entrance side to replace task *i* without the violation of cycle time constraint and precedence relationship constraint. For a given partial solution, a sub-problem containing this partial solution is pruned if 1) there is a task *i* assigned to the exit side of the last station and an unallocated task *h* such that task *i* and task *h* have no precedence relationship, $t_i \leq t_h$ and $P_i^* \subseteq P_h^*$, and 2) task *h* can be assigned to the exit side to replace task *i* without the violation of cycle time constraint and precedence relationship constraint.

The no-successors and no-predecessors rule: The current sub-problem is pruned if 1) the tasks on the entrance side of the last station in a partial solution have no successors, 2) the tasks on the exit side of the last station in a partial solution have no predecessors, 3) and there exists an unassigned task which can be allocated to the entrance (exit) side and has at least one successor (predecessor).

Clearly, the maximum load rule will never prevent the discovery of the optimal solution. The memory-based dominance rule is applied to two already generated sub-problems to preserve one sub-problem with equivalent or smaller station number, and thus memory-based dominance rule will never prune an optimal solution. The only remaining dominance rules are modified extended Jackson rule and the no-successors and no-predecessors rule, and the correctness and the compatibility of these two dominance rules are discussed in Appendix A.

3.6 Search strategy

The order in which sub-problem is selected to be explored has a great impact on the computational time of a branch and bound algorithm. In the literature, there are four main search strategies to determine this order: depth-first search strategy (DFS) [8], best-first search strategy (BFS) [17], BrFS and CBFS [11, 12].

There are variations of DFS strategies [5-9], and among them the SALOME [8, 9] exhibits the best performance in solving SALBP. BFS selects the most promising sub-problem from a set of sub-problems. For instance, Yolmeh and Salehi [17] selects the sub-problem with a higher number of stations or the sub-problem with lower generation based lower bound. The sub-problem selection method has impact on the overall speed of achieving a complete solution, and it is demonstrated that the BPR with BFS strategy outperforms the ULINO with DFS in solving UALBP. BrFS strategy

generates all the sub-problems at depth 1, and then at depth 2, and this procedure is repeated until the optimal solution is found. BrFS is heavy and it might not achieve a complete solution within a time limit, but BrFS could be utilized to prove the optimality of some special cases.

CBFS strategy is a hybrid method between DFS and BFS. BBR with CBFS produces the state-of-the-art results for SALBP. In this method, CBFS selects the most promising sub-problem at depth 1 and generates a set of new children for depth 2, and then it selects the promising sub-problem at depth 2 and generates a set of new children for depth 3. This procedure is terminated upon the deepest level is reached and then it comes back to depth 1 and this cycle is repeated. Regarding the selection of the sub-problem, they define $b(\wp) = I/m - \lambda|U|$, where $\wp = (A, U, S_1, S_2, \cdots, S_m)$, I is the total idle in the former *m* stations and λ is an input parameter set to 0.02. The sub-problem with minimum value of measure $b(*)$ is selected to be explored.

Due to the superiority of CBFS in SALBP, this research also extends CBFS to UALBP with two improvements, denoted as modified CBFS, as follows. Firstly, the sub-problem at depth l is not selected when there are plenty of sub-problems (set to 10,000) at depth l+1. Secondly, this research employs a new sub-problem selection indicator by hybridizing that with Sewell and Jacobson [11] and Yolmeh and Salehi [17] as shown in Equation (5). In this equation, LB(U) is the lower bound of the unassigned tasks obtained using LB1, LB2, LB3 and BPLB. This expression ensures that the sub-problem with smaller lower bound is selected primarily and the sub-problem with less idle time on the former *m* stations is selected secondarily and the larger number of remained tasks is preferred when they have the same lower bound.

$$b(\wp) = \mathrm{LB}(U) - 10 \cdot \mathrm{m} + I/m - \lambda|U| \quad (5)$$

Algorithm 3: Modified CBFS

Initialize level 0 by storing the root problem; *l*:=1;
While (unexplored sub-problems exist)
  While (the number of sub-problems at depth *l*+1≥10,000)
    l := *l*+1;
  Endwhile
  Select a sub-problem at depth l with the minimum value of $b(*)$;
  Store all the non-dominated children of this selected best sub-problem at depth *l*+1;
  l := *l* % (UB-1);
Endwhile

The first improvement is applied to attempt to accelerate the search process. In our initial experiments, there are too many sub-problems in the former depths and only almost $1/(UB-1)$ time is utilized at one latter depth. This modification ensures that more computational effort is applied for latter depths to achieve the optimal solution fast while preserving enough sub-problems in the former depths. The second improvement is inspired by that the sub-problems in the same depth might have different lower bounds when solving large-size instances. Clearly, the smaller lower bound has higher chance to find better UB and hence increases the search speed by pruning more sub-problems.

4. Computational study

This section utilizes two benchmark datasets to test the performance of the proposed methodology: the well-known 269 benchmark problems utilized by Scholl and Klein [16] and the new data set provided

by Otto and Otto [22] in 2013. The new data set is divided into four types of problems: small-size problem set with 20 tasks, medium-size problem set with 50 tasks, large-size problem set with 100 tasks and very large-size problem set with 1000 tasks. There are 525, 5250, 525 and 525 instances in small-size, medium-size, large-size and very large-size problem sets, respectively. These rich benchmark datasets provide the opportunity of observing the performance of the tested algorithms for various size instances, and also make the achieved findings more convincing.

To observe the performance of the proposed improvements in the previous section, this section implements several variants of the BBR algorithm as specified in Table 1. The first column refers to the abbreviation of the tested algorithms, where appropriate suffixes are used to highlight the search strategy, dominance rule and the sequence of the tasks in enumeration employed by each BBR method. Note that BBR_CNP refers to the proposed BBR method.

Table 1 Tested BBR algorithms

Algorithms	Search strategy	Dominance rule	The sequence of the tasks in enumeration
BBR_BSD	BFS	Dominance rule in Scholl and Klein [16]	Don't renumber the tasks
BBR_BND	BFS	New dominance rule	Don't renumber the tasks
BBR_CSD	CBFS	Dominance rule in Scholl and Klein [16]	Don't renumber the tasks
BBR_CND	CBFS	New dominance rule	Don't renumber the tasks
BBR_BNP	BFS	New dominance rule	Renumber the tasks using positional weight and operation time
BBR_BNO	BFS	New dominance rule	Renumber the tasks using operation time
BBR_CNP	CBFS	New dominance rule	Renumber the tasks using positional weight and operation time
BBR_CNO	CBFS	New dominance rule	Renumber the tasks using operation time

All the implemented algorithms utilize backtracking rule to save memory and propose the MHH to achieve upper bound in Phase I, where the total number of generated loads in one station is limited to 1,000. Following Sewell and Jacobson [11], the number of generated full loads from one sub-problem in Phase II is limited to 10,000 to avoid tremendous time in selecting a load and much memory to store the loads for one station. In Phase II, all the algorithms select the sub-problem $\wp$ with the smallest value of $b(\wp)$. Specifically, all the algorithms utilizing cyclic best-first limit the number of sub-problems in a depth up to 1,000 and the sub-problems at a depth is not selected for search when there are greater than or equal to 10,000 sub-problems at a latter depth.

These tested methodologies are compared with other two exact methods: ULINO in Scholl and Klein [16] and BPR in Yolmeh and Salehi [17], where the results by these two methods are taken from the published papers directly. Recall that ULINO and BPR are the only applied exact methods for U-shaped assembly line balancing, and BPR is only method which has solved the new data set in 2013. The tested algorithms are implemented in Microsoft Visual Studio 2015, and the experiments are run on a personal computer with Intel(R) Core(TM) i7-4790S 3.20GHZ CPU and 8.00 GB of available memory. All these methods terminate when the optimal solution is achieved or the computation time reaches to 500 seconds as in Scholl and Klein [16] and Yolmeh and Salehi [17].

4.1 Experimental results on Scholl data set

This section evaluates the performances of the algorithms on Scholl data set and reports the achieved results in Table 2 (see Appendix B for the detailed results). In Table 2, the second column (*#OPT found*)

denotes the number of achieved optimal solutions, among which some might not be proved optimal by the corresponding algorithm. The third column (*#OPT verified*) gives the number of achieved optimal solutions which have been proved to be optimal by the corresponding algorithm. The relative percentage deviation (RPD) is also employed to measure the gap between the LB and the achieved upper bound (UP) using Equation (6).

$$RPD = 100 \times (UP - LB)/LB \quad (6)$$

The average relative percentage deviation (*RPD-Avg*) and the maximum relative percentage deviation (*RPD-Max*) are presented in the fourth and fifth columns, respectively. The last three columns compare the tested algorithms with ULINO: the number of cases when the algorithm achieves better solution than ULINO (*Better than ULINO*), same solution to ULINO (*Same to ULINO*), and worse solution than ULINO (*Worse than ULINO*).

From Table 2, it is observed that ULINO is the worst performer, which finds 238 optimal solutions, and BPR is the second worst performer which finds 255 optimal solutions. On the contrary, all the implemented BBR algorithms produce better results, among which the BBR_CNP is the best performer with 259 optimal solutions. BBR_BNP and BBR_BNO are the second-best performers with 258 achieved optimal solutions. Regarding the RPD-Avg, BBR_CNP is also the best performer and BBR_BNP and BBR_BNO are the second-best performers.

Table 3 presents the CPU times by these algorithms when solving all the instances, where the average value, the standard deviation, the minimum value and the maximum value are presented in the second, third, fourth and fifth columns, respectively. Clearly, BBR_CNP consumes the smallest average running time of 20.38s with the smallest standard deviation of 97.62, and BBR_BNP consumes the second smallest average running time of 22.73s with the second smallest standard deviation of 102.83. Among the remaining methods, ULINO consumes the largest average running time and BPR has the second largest average running time.

Table 2 Overall results obtained by the algorithms

Algorithm	#OPT found	#OPT verified	RPD-Avg	RPD-Max	Better than ULINO	Same to ULINO	Worse than ULINO
ULINO	238	233	0.52	10.00	-	-	-
BPR	255	255	0.26	7.14	17	252	0
BBR_BSD	257	257	0.22	6.67	19	250	0
BBR_BND	257	257	0.22	6.67	19	250	0
BBR_CSD	257	257	0.22	6.67	19	250	0
BBR_CND	257	257	0.22	6.67	19	250	0
BBR_BNP	258	258	0.20	7.69	20	249	0
BBR_BNO	258	258	0.20	7.69	20	249	0
BBR_CNP	**259**	**259**	**0.18**	7.69	**21**	248	0
BBR_CNO	257	257	0.21	7.69	19	250	0

*Best in bold.

Table 3 Running times by algorithms

Algorithm	Average (s)	Standard deviation	Minimum (s)	Maximum(s)
ULINO	82.09	-	-	-
BPR	34.66	120.79	0.01	815.07
BBR_BSD	23.11	103.59	0.00	501.01
BBR_BND	23.01	103.60	0.00	504.38
BBR_CSD	24.62	106.26	0.00	501.00
BBR_CND	23.92	104.70	0.00	525.15
BBR_BNP	22.73	102.83	0.00	634.39
BBR_BNO	24.48	106.89	0.00	603.93
BBR_CNP	**20.38**	**97.62**	0.00	608.70
BBR_CNO	23.50	105.02	0.00	577.59

*Best in bold.

Recall that some maximums of running times in Table 3 exceed 500 seconds. This is because the termination criterion is checked after generating 10,000 station loads for a selected sub-problem, as explained in Section 3.1. Also, the consumed time to utilize the dominance rules or calculate the lower bounds might be large, especially when BPLB is used. Hence, it is normal that the running time is larger than the given time limit, which is 500 seconds. Recall that, if we check whether the termination criterion is satisfied after generating each station load, a lot of time will be wasted. Still, for most cases that are not solved optimally, the running time is close to 500 seconds and there are only few special cases with more than 600 seconds of running time.

Table 4 presents the detailed results on challenging instances, which ULINO is not capable to find the optimal solution. It is observed that BPR and BBR algorithms achieve many better solutions. As presented in Table 2, BPR, BBR_BSD, BBR_BND, BBR_CSD, BBR_CND, BBR_BNP, BBR_BNO, BBR_CNP and BBR_CNO outperform ULINO in 17, 19, 19, 19, 19, 20, 20, 21 and 19 cases, respectively. Specifically, for Arcus 2 with 111 tasks and the cycle time of 8356, only BBR_CNP achieves the optimal solution. For Arcus 2 with 111 tasks and the cycle time of 9400 and 10027, BBR_BNP, BBR_BNO, BBR_CNP and BBR_CNO achieve the optimal solutions by renumbering the tasks. Nevertheless, task renumbering produces worse results when solving Arcus 2 with 111 tasks and the cycle time of 11570.

Table 4 Results on challenging instances

Instance	No. of tasks	Cycle time	LB	ULINO	BPR	BBR_BSD	BBR_BND	BBR_CSD	BBR_CND	BBR_BNP	BBR_BNO	BBR_CNP	BBR_CNO
				UB	UB	UB	UB	UB	UB	UB	UB	UB	UB
Kilbridge	45	56	10	11	**10**	**10**	**10**	**10**	**10**	**10**	**10**	**10**	**10**
Warnecke	58	54	31	31	**31**	**31**	**31**	**31**	**31**	**31**	**31**	**31**	**31**
		62	26	27	**26**	**26**	**26**	**26**	**26**	**26**	**26**	**26**	**26**
		65	25	25	**25**	**25**	**25**	**25**	**25**	**25**	**25**	**25**	**25**
		68	23	24	**23**	**23**	**23**	**23**	**23**	**23**	**23**	**23**	**23**
		71	22	23	**22**	**22**	**22**	**22**	**22**	**22**	**22**	**22**	**22**
		82	19	20	**19**	**19**	**19**	**19**	**19**	**19**	**19**	**19**	**19**
Tonge	70	160	22	23	**22**	**22**	**22**	**22**	**22**	**22**	**22**	**22**	**22**
Tonge		176	20	21	**20**	**20**	**20**	**20**	**20**	**20**	**20**	**20**	**20**
Wee-mag	75	47	32	33	**32**	**32**	**32**	**32**	**32**	**32**	**32**	**32**	**32**
Arcus 1	83	3786	21	22	**21**	**21**	**21**	**21**	**21**	**21**	**21**	**21**	**21**
Mukherje	94	176	24	25	**24**	**24**	**24**	**24**	**24**	**24**	**24**	**24**	**24**
Arcus 2	111	5785	26	27	27	27	27	27	27	27	27	27	27
		6016	25	26	26	26	26	26	26	26	26	26	26
		6267	24	25	25	25	25	25	25	25	25	25	25
		6540	23	24	24	24	24	24	24	24	24	24	24
		6837	22	23	23	23	23	23	23	23	23	23	23
		7162	21	22	22	22	22	22	22	22	22	22	22
		7520	20	21	21	21	21	21	21	21	21	21	21
		7916	19	20	20	20	20	20	20	20	20	20	20
		8356	18	19	19	19	19	19	19	19	19	**18**	19
		8847	17	18	18	18	18	18	18	18	18	18	18
		9400	16	17	17	17	17	17	17	**16**	**16**	**16**	**16**
		10027	15	16	16	16	16	16	16	**15**	**15**	**15**	**15**
		10743	14	15	15	**14**	**14**	**14**	**14**	**14**	**14**	**14**	**14**
		11570	13	14	**13**	**13**	**13**	**13**	**13**	14	14	14	14
Barthol2	148	85	50	51	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**
		89	48	49	**48**	**48**	**48**	**48**	**48**	**48**	**48**	**48**	**48**

		93	46	47	**46**	**46**	**46**	**46**	**46**	**46**	**46**	**46**	**46**
		97	44	45	**44**	**44**	**44**	**44**	**44**	**44**	**44**	**44**	**44**
Scholl	297	1394	50	51	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**
		1422	49	50	50	**49**	**49**	**49**	**49**	**49**	**49**	**49**	50
		1515	46	47	**46**	**46**	**46**	**46**	**46**	**46**	**46**	**46**	**46**

*Better than ULINO in bold.

To sum up, all the implemented BBR methods produce promising results by outperforming ULINO and BPR. BBR_CNP is the best performer among these methods by producing slightly better results. Furthermore, renumbering the tasks helps the achievement of three optimal solutions with the cost of losing one optimal solution. This proves that renumbering the tasks is a good option for some unresolved challenging instances.

4.2 Experimental results on new data set

This section presents the results of the new dataset in Table 5 (see Appendix B for the detailed results), where the results by MHH are also included. In this table, *#instances* column denotes the number of instances, *#OPT found* denotes the number of achieved optimal solutions, *#OPT verified* is the number of achieved optimal solutions proved to be optimal, and *RPD-Avg* and *RPD-Max* are the average and maximum of RPD values.

Table 5 Overall results on the new data set

Method		Instance				
		Small-size	Medium-size	Large-size	Very large-size	Combined
	#instances	525	5250	525	525	6825
MHH	#OPT found	489	4126	357	336	5308
	#OPT verified	-	-	-	-	-
	RPD-Avg	0.69	1.06	1.43	1.93	1.12
	RPD-Max	20.00	14.29	12.96	10.28	20.00
BPR	#OPT found	525	5241	481	346	6593
	#OPT verified	525	5236	478	346	6585
	RPD-Avg	0.00	0.01	0.26	1.72	0.16
	RPD-Max	0.00	4.35	5.77	9.88	9.88
BBR_BSD	#OPT found	525	5244	509	342	6620
	#OPT verified	525	5243	509	343	6620
	RPD-Avg	0.00	0.00	0.06	1.69	0.14
	RPD-Max	0.00	4.35	3.70	10.28	10.28
BBR_BND	#OPT found	525	5244	508	349	6626
	#OPT verified	525	5243	508	349	6625
	RPD-Avg	0.00	0.00	0.07	1.69	0.14
	RPD-Max	0.00	4.35	3.85	10.28	10.28
BBR_CSD	#OPT found	525	5245	510	343	6623
	#OPT verified	525	5245	510	343	6623
	RPD-Avg	0.00	0.00	0.06	1.19	0.10
	RPD-Max	0.00	4.35	3.70	7.31	7.31
BBR_CND	#OPT found	525	5245	510	349	6629
	#OPT verified	525	5244	510	349	6628
	RPD-Avg	0.00	0.00	0.06	1.18	0.10
	RPD-Max	0.00	4.35	3.70	7.51	7.51
BBR_BNP	#OPT found	525	5244	510	350	6629
	#OPT verified	525	5243	510	350	6628
	RPD-Avg	0.00	0.00	0.07	1.66	0.14
	RPD-Max	0.00	4.35	3.85	10.08	10.08
BBR_BNO	#OPT found	525	5244	510	347	6626
	#OPT verified	525	5243	509	347	6624
	RPD-Avg	0.00	0.00	0.07	1.65	0.14
	RPD-Max	0.00	4.35	3.85	9.88	9.88
BBR_CNP	#OPT found	525	5245	510	**350**	**6630**
	#OPT verified	525	5244	510	**350**	**6629**
	RPD-Avg	0.00	0.00	0.06	1.16	0.10
	RPD-Max	0.00	4.35	3.85	7.51	7.51

BBR_CNO	#OPT found	525	5245	510	347	6627
	#OPT verified	525	5244	510	347	6626
	RPD-Avg	0.00	0.00	0.06	**1.12**	**0.09**
	RPD-Max	0.00	4.35	3.70	7.71	7.71

*Best in bold.

When observing the optimal solutions found in each set of instances (small-size, medium-size, large-size and very large-size), BPR has the same performance when solving small-size instances, slightly worse performance when solving medium-size instances, and much worse performance when solving large-size instances. Regarding the very large-size instances, BPR shows slightly better performance over BBR_BSD and BBR_CSD, and slightly worse performance than BBR_BND, BBR_CND, BBR_BNP, BBR_BNO, BBR_CNP and BBR_CNO. Regarding the overall achieved optimal solutions, MHH is the worst performer which is reasonable as MHH is only applied to achieve the upper bound for BBR algorithms. Still, MHH is quite effective heuristic method with 5308 optimal solutions, which is equivalent to over 77% of all tested instances. BPR is the second worst performer with 6593 found optimal solutions, and again all the implemented BBR algorithms produce better results. Specifically, BBR_CNP is the best performer with 6630 found optimal solutions, equivalent to the optimal solutions for over 97% of the test instances. BBR_CND and BBR_BNP are the second-best performers with 6629 found optimal solutions. With regard to overall average RPD, BBR_CNO is the best performer with an RPD-Avg of 0.09, and BBR_CNP is the second-best performer with an RPD-Avg of 0.10.

Table 6 exhibits the comparison between BPR and BBR methods, where *Better than BPR*, *Same to BPR* and *Worse than BPR* denote the number of cases for which the corresponding BBR method produces better, the same or worse results. Regarding the overall number of Better than BPR, BBR_CNO is the best one with 221 better solutions, but it also obtains 10 worse solutions. BBR_CNP is the second best one with 217 better solutions and 8 worse solutions.

Table 6 Comparison between BPR and BBR methods

Method		Instance				
		Small-size	Medium-size	Large-size	Very large-size	Combined
	#instances	525	5250	525	525	6825
BBR_BSD	Better than BPR	0	4	40	80	124
	Same to BPR	525	5245	481	357	6608
	Worse than BPR	0	1	4	88	93
BBR_BND	Better than BPR	0	4	38	77	119
	Same to BPR	525	5245	483	361	6614
	Worse than BPR	0	1	4	87	92
BBR_CSD	Better than BPR	0	4	40	175	219
	Same to BPR	525	5246	481	342	6594
	Worse than BPR	0	0	4	8	12
BBR_CND	Better than BPR	0	4	40	174	218
	Same to BPR	525	5246	481	348	6600
	Worse than BPR	0	0	4	3	7
BBR_BNP	Better than BPR	0	4	39	78	121
	Same to BPR	525	5245	482	359	6611
	Worse than BPR	0	1	4	88	93
BBR_BNO	Better than BPR	0	4	39	83	126
	Same to BPR	525	5245	482	356	6608
	Worse than BPR	0	1	4	86	91
BBR_CNP	Better than BPR	0	4	41	172	217
	Same to BPR	525	5246	479	350	6600
	Worse than BPR	0	0	5	3	8
BBR_CNO	Better than BPR	0	4	41	**176**	**221**
	Same to BPR	525	5246	479	344	6594
	Worse than BPR	0	0	5	5	10

*Best in bold.

The BBR methods using CBFS (BBR_CSD, BBR_CND, BBR_CNP and BBR_CNO) outperform those using BFS (BBR_BSD, BBR_BND, BBR_BNP and BBR_BNO) by achieving more than 200 better results. In fact, the BBR methods with BFS or CBFS have similar performance when solving small-size, medium-size and large-size instances, whereas the applied search strategy distinguishes the performance of the implemented BBR methods clearly when solving the very large-size instances. This finding suggests that CBFS is more effective than BFS when solving very large-size instances. When comparing the value of Better than BPR and Worse than BPR values in the Combined column, the Better than BPR values are larger than Worse than BPR for all BBR methods, demonstrating the superiority of the proposed BBR methods over BPR.

To evaluate the running times of the implemented methods, Table 7 presents the computational times for BPR and BBR algorithms. While the BBR methods consume more time than BPR when solving small-size cases, they consume much less time than BPR when solving medium-size, large-size and very large-size cases. BBR_CNP is the fastest methodology for large-size and very large-size instances and it has the smallest overall average running time. BBR_CND and BBR_BNP have the second and third smallest overall average running times, respectively. All the BBR algorithms have similar running time and the difference in the overall average running times is lower than one second. Nevertheless, the difference between BPR and BBR methods in the overall average running time is more than 10 seconds, and the proposed BBR algorithms are clearly faster to achieve optimal solutions.

Table 7 Running times on new dataset

Method		Instance				
		Small-size	Medium-size	Large-size	Very large-size	Combined
	#instances	525	5250	525	525	6825
BPR	Average (s)	0.16	3.54	60.54	253.99	26.93
	Standard deviation	0.22	29.32	146.63	188.54	97.83
	Minimum (s)	0.01	0.01	0.11	15.30	0.01
	Maximum (s)	2.22	501.24	509.82	532.69	532.69
BBR_BSD	Average (s)	0.24	0.99	20.79	175.85	15.90
	Standard deviation	0.38	18.30	93.74	238.66	86.41
	Minimum (s)	0.00	0.00	0.00	0.01	0.00
	Maximum (s)	1.42	501.38	502.62	501.90	502.62
BBR_BND	Average (s)	0.23	0.99	21.64	169.31	15.47
	Standard deviation	0.37	18.30	96.57	235.99	85.16
	Minimum (s)	0.00	0.00	0.00	0.02	0.00
	Maximum (s)	1.50	501.62	503.78	501.28	503.78
BBR_CSD	Average (s)	0.22	0.86	19.01	175.45	15.64
	Standard deviation	0.34	16.91	89.12	238.29	85.69
	Minimum (s)	0.00	0.00	0.00	0.02	0.00
	Maximum (s)	0.99	501.40	501.41	502.44	502.44
BBR_CND	Average (s)	0.21	0.86	19.19	170.74	15.29
	Standard deviation	0.33	16.94	89.00	235.92	84.50
	Minimum (s)	0.00	0.00	0.00	0.01	0.00
	Maximum (s)	0.96	501.38	501.37	502.06	502.06
BBR_BNP	Average (s)	0.23	0.99	20.21	168.56	15.30
	Standard deviation	0.37	18.30	92.64	235.52	84.62
	Minimum (s)	0.00	0.00	0.00	0.02	0.00
	Maximum (s)	1.49	501.53	501.38	501.27	501.53
BBR_BNO	Average (s)	0.23	1.00	20.34	171.04	15.51
	Standard deviation	0.37	18.30	93.52	236.74	85.30
	Minimum (s)	0.00	0.00	0.00	0.02	0.00
	Maximum (s)	1.48	501.66	502.51	501.19	502.51
BBR_CNP	Average (s)	0.22	0.86	**18.87**	**169.99**	**15.21**
	Standard deviation	0.35	16.94	88.58	235.40	84.25
	Minimum (s)	0.00	0.00	0.00	0.01	0.00
	Maximum (s)	1.06	501.52	501.39	502.00	502.00

BBR_CNO	Average (s)	0.21	0.87	19.23	172.29	15.41
	Standard deviation	0.34	16.94	89.72	236.67	84.94
	Minimum (s)	0.00	0.00	0.00	0.01	0.00
	Maximum (s)	1.11	501.45	502.00	502.10	502.10

*Best average time in bold.

Table 8 exhibits the newly found optimal solutions or newly proved optimal solutions by BBR methods. All the instances have a cycle time fixed to 1,000. For the medium-size instance with 50 tasks, three optimal solutions are found by BBR methods for the first time. Moreover, five solutions are proved to be optimal for the first time. For large-size instances with 100 tasks, BBR methods achieve 36 new optimal solutions and prove the optimality for three cases for the first time.

Table 8 Newly found or proved optimal solutions by implemented methods

Instance	LB	BPR	BBR_BSD	BBR_BND	BBR_CSD	BBR_CND	BBR_BNP	BBR_BNO	BBR_CNP	BBR_CNO
		UB	UB	UB	UB	UB	UB	UB	UB	UB
instance_n=50_255p3	28	28	28	28	28	28	28	28	28	28
instance_n=50_254p9	30	31	**30**	**30**	**30**	**30**	**30**	**30**	**30**	**30**
instance_n=50_115	26	27	**26**	**26**	**26**	**26**	**26**	**26**	**26**	**26**
instance_n=50_107p9	27	28	**27**	**27**	**27**	**27**	**27**	**27**	**27**	**27**
instance_n=50_107p8	28	28	28	28	28	28	28	28	28	28
instance_n=50_107p5	28	28	28	28	28	28	28	28	28	28
instance_n=50_105p9	24	24	24	24	24	24	24	24	24	24
instance_n=50_104p6	26	26	26	26	26	26	26	26	26	26
instance_n=100_525	55	58	**55**	**55**	**55**	**55**	**55**	**55**	**55**	**55**
instance_n=100_524	53	55	**53**	**53**	**53**	**53**	**53**	**53**	**53**	**53**
instance_n=100_523	52	53	**52**	**52**	**52**	**52**	**52**	**52**	**52**	**52**
instance_n=100_522	52	55	**52**	**52**	**52**	**52**	**52**	**52**	**52**	**52**
instance_n=100_520	54	55	**54**	**54**	**54**	**54**	**54**	**54**	**54**	**54**
instance_n=100_519	57	58	**57**	**57**	**57**	**57**	**57**	**57**	**57**	**57**
instance_n=100_518	52	54	**52**	**53**	**52**	**52**	**52**	**52**	**52**	**52**
instance_n=100_515	57	58	**57**	**57**	**57**	**57**	**57**	**57**	**57**	**57**
instance_n=100_514	52	54	**52**	**52**	**52**	**52**	**52**	**52**	**52**	**52**
instance_n=100_513	54	57	**54**	**54**	**54**	**54**	**54**	**54**	**54**	**54**
instance_n=100_512	58	59	**58**	**58**	**58**	**58**	**58**	**58**	**58**	**58**
instance_n=100_510	52	54	**52**	**52**	**52**	**52**	**52**	**52**	**52**	**52**
instance_n=100_509	53	55	**53**	**53**	**53**	**53**	**53**	**53**	**53**	**53**
instance_n=100_508	52	55	**52**	**52**	**52**	**52**	**52**	**52**	**52**	**52**
instance_n=100_507	55	55	55	55	55	55	55	55	55	55
instance_n=100_506	53	55	**53**	**53**	**53**	**53**	**53**	**53**	**53**	**53**
instance_n=100_505	54	57	**54**	**54**	**54**	**54**	**54**	**54**	**54**	**54**
instance_n=100_504	55	58	56	56	**55**	**55**	56	56	**55**	**55**
instance_n=100_503	57	58	**57**	**57**	**57**	**57**	**57**	**57**	**57**	**57**
instance_n=100_502	61	62	**61**	**61**	**61**	**61**	**61**	**61**	**61**	**61**
instance_n=100_501	59	60	**59**	**59**	**59**	**59**	**59**	**59**	**59**	**59**
instance_n=100_472	23	23	23	23	23	23	23	23	23	23
instance_n=100_463	25	25	25	25	25	25	25	25	25	25
instance_n=100_450	52	53	53	53	**52**	**52**	53	53	**52**	**52**
instance_n=100_448	54	55	**54**	**54**	**54**	**54**	**54**	**54**	**54**	**54**
instance_n=100_445	54	55	**54**	**54**	**54**	**54**	**54**	**54**	**54**	**54**
instance_n=100_440	51	52	**51**	**51**	**51**	**51**	**51**	**51**	**51**	**51**
instance_n=100_438	52	53	**52**	**52**	**52**	**52**	**52**	**52**	**52**	**52**
instance_n=100_436	49	50	**49**	**49**	**49**	**49**	**49**	**49**	**49**	**49**
instance_n=100_426	58	60	**58**	**58**	59	59	**58**	**58**	59	59
instance_n=100_296	53	54	54	54	54	54	**53**	**53**	**53**	**53**
instance_n=100_294	54	55	**54**	**54**	**54**	**54**	**54**	**54**	**54**	**54**
instance_n=100_280	51	53	**52**	**52**	**51**	**51**	**52**	**52**	**51**	**51**
instance_n=100_148	51	52	**51**	**51**	**51**	**51**	**51**	**51**	**51**	**51**
instance_n=100_144	47	48	**47**	**47**	**47**	**47**	**47**	**47**	**47**	**47**

instance_n=100_143	51	52	**51**	**51**	**51**	**51**	**51**	**51**	**51**	**51**
instance_n=100_135	53	54	**53**	**53**	**54**	**54**	**53**	**53**	**54**	**54**
instance_n=100_130	53	54	**53**	**53**	**53**	**53**	**53**	**53**	**53**	**53**
instance_n=100_126	50	52	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**
instance_n=1000_61	229	230	230	**229**	**230**	**229**	**229**	**229**	**229**	**229**
instance_n=1000_516	229	230	**229**	**229**	**229**	**229**	**229**	**229**	**229**	**229**
instance_n=1000_503	224	225	**224**	**224**	**224**	**224**	**224**	**224**	**224**	**224**
instance_n=1000_278	220	221	**220**	**220**	**220**	**220**	**220**	**220**	**220**	**220**

Regarding the very large-size instance with 1,000 tasks, BBR methods achieve four new optimal solutions. Among all the BBR methods, BBR_BSD and BBR_BND achieve 38 new optimal solutions, BBR_CSD achieves 39 new optimal solutions; BBR_CND, BBR_BNP and BBR_BNO achieve 40 new optimal solutions; and BBR_CNP and BBR_CNO achieve 41 new optimal solutions. Clearly, BBR_CNP and BBR_CNO are the best performers regarding the number of newly found optimal solutions.

In summary, all the BBR methods are capable to achieve better results than the published BPR. BBR_CNP is the best performer regarding the number of achieved optimal solutions. CBFS outperforms the BFS by a significant margin when solving very large-size instances. Again, renumbering the tasks helps achieve more optimal solutions than the lost optimal solutions, indicating that renumbering the tasks is worth trying and shows slightly better performance.

5. Conclusions and future research

This research presented a BBR algorithm to solve UALBP. The BBR algorithm applies the CBFS as the search strategy and remembers all the searched sub-problems to eliminate redundant sub-problems. It also proposes several improvements in solving large-size problems: two new dominance rules, renumbering the tasks when generating the station loads, new criterion to select the most promising sub-problem and limiting the number of sub-problems at each depth. A comprehensive study has been carried out where eight BBR methods with different configurations were evaluated and compared with the two current best exact algorithms on Scholl's 269 benchmark instances and the new 6825 instances. Computational results demonstrated that the tested BBR methodologies outperform the ULINO and BPR algorithms by achieving many new optimal solutions or upper bounds. The proposed BBR method achieved 259 optimal solutions for Scholl's well-known 269 benchmark problems, where ULINO and BPR algorithm found 238 and 255 optimal solutions, respectively. The proposed method achieved optimality for slightly over 97% of the problems in the new dataset where BPR algorithm achieves the optimality for over 96% of tested problems. Specifically, one Scholl's instance (Scholl 297 with a cycle time of 1422), whose optimal solution was not known for almost 20 years, is solved optimally for the first time. The proposed BBR method also finds 41 new optimal solutions for the new dataset.

As the search space of UALBP is much larger and there are many sub-problems at each depth, future research might focus on developing various pruning rules and dominance rules to reduce the search space. Another problem when utilizing exact methods is that they might terminate due to out of memory problems in solving large-size instances, and it is interesting to develop methods to optimize memory usage and reduce the utilized computer memory to store the sub-problems. Another approach is testing the most promising station loads while limiting the number of generated station loads, which can also help increase the search speed. As many optimization objectives are involved in real world systems, future researches might extend the proposed exact methods or hybrid exact methods with multi-objective metaheuristics to solve multi-objective optimization problems. Another research avenue is extending the proposed methods to other variants of assembly line balancing problems,

including assembly line balancing with sequence-dependent setup times, parallel assembly line balancing, two-sided assembly line balancing, and robotic assembly line balancing.

Appendix A. The correctness and compatibility of the dominance rules

Lemma 1. For a given partial solution, a sub-problem containing this partial solution can be pruned if 1) there is a task *i* assigned to the entrance side of the last station and an unallocated task *h* such that task *i* and task *h* have no precedence relationship, $t_i \le t_h$ and $P_i^* \subseteq P_h^*$, and 2) task *h* can be assigned to the entrance side to replace task *i* without violation of cycle time constraint and precedence relationship constraint. For a given partial solution, a sub-problem containing this partial solution can be pruned if 1) there is a task *i* assigned to the exit side of the last station and an unallocated task *h* such that task *i* and task *h* have no precedence relationship, $t_i \le t_h$ and $F_i^* \subseteq F_h^*$, and 2) task *h* can be assigned to the exit side to replace task *i* without the violation of cycle time constraint and precedence relationship constraint.

Proof 1. Let $X = S_1 \cup S_2 \cup \cdots \cup S_j \cup S_{j+1} \cup \cdots$, $i \in S_j$ and $h \in S_{j+1} \cup \cdots$ be a feasible solution. There must be one solution by exchanging the position of task *i* and task *h*: $Y = S_1 \cup S_2 \cup \cdots \cup S_j \cup S_{j+1} \cup \cdots$, $h \in S_j$ and $i \in S_{j+1} \cup \cdots$ when 1) task *i* is assigned to the entrance side, task *i* and task *h* have no precedence relationship, $t_i \le t_h$ and $F_i^* \subseteq F_h^*$, and 2) task *h* can be assigned to the entrance side to replace task *i* without the violation of cycle time constraint and precedence relationship constraint. The two solutions *X* and *Y* have the same task assignment except the exchanged position of task *i* and task *h*, and clearly the two solutions have the same station number. The same situation also applies to the backward direction. Hence, deleting the solution X by pruning its sub-problem using the modified extended Jackson rule will not prevent the discovery of the optimal solution.

Lemma 2. The current sub-problem can be pruned if 1) the tasks on the entrance side of the last station in a partial solution have no successors, 2) the tasks on the exit side of the last station in a partial solution have no successors, 3) and there exists an unassigned task which can be allocated to the entrance side and has at least one successor or can be allocated to the exit side and has at least one predecessor.

Proof 2. Let $X = S_1 \cup S_2 \cup \cdots \cup S_j \cdots \cup S_k \cup \cdots$ be a feasible solution, FS_j and BS_j are the task sets on the entrance side and exit side of station *j* and FS_k and BS_k are the task sets on the entrance side and exit side of station *k* in this solution. There must be one solution by exchanging the position of task set FS_j and BS_j with task set FS_k and BS_k: $Y = S_1 \cup S_2 \cup \cdots \cup (FS_k \cup BS_k) \cup \cdots \cup (FS_j \cup BS_j) \cup \cdots$ if 1) the tasks in FS_j have no successors, 2) the tasks in BS_j have no predecessors, 3) and there exists an unassigned task in FS_k with at least one successor or in BS_k with at least one predecessor. The two solutions *X* and *Y* have the same task assignment except the exchanged position of tasks in station *j* and station *k*, and clearly the two solutions have the same station number. Hence, deleting the solution X by pruning its sub-problem using no-successors and no-predecessors rule will not prevent the discovery of the optimal solution.

The compatibility of the rules is studied here, and clearly there is no conflict between maximum load rule and others and between memory-based dominance rule and others (see Morrison, Sewell [12] for details). Hence, this section focuses on the compatibility of modified extended Jackson rule (MEJR) and no-successors and no-predecessors rule (NSPR) following Morrison, Sewell [12].

Lemma 3. Let X, Y and Z be three sub-problems in one search tree. If X is dominated by Y using

MEJR and Y is dominated by Z using NSPR, X is also dominated by Z using NSPR.

Proof. Let $X=(A,U,S_1,S_2,\cdots,S_{m-1},S_m)$, $Y=(A',U',S_1,S_2,\cdots,S_{m-1},S_m')$ and X is dominated by Y using MEJR. There must be $i\in S_m$, $h\in S_m'$, $t_h\geq t_i$, $F_h^*\supseteq F_i^*$ and $S_m'=(S_m-\{i\})\cup\{h\}$ when both task i and task h are allocated to the entrance side. Otherwise, there must be $i\in S_m$, $h\in S_m'$, $t_h\geq t_i$, $P_h^*\supseteq P_i^*$ and $S_m'=(S_m-\{i\})\cup\{h\}$ when both task i and task h are allocated to the exit side. If these two tasks are allocated to the entrance side, $F_h^*=\phi$ and hence $F_i^*=\phi$ as well. Then, the tasks in S_m also have no successor and X is dominated by Z using NSPR. If these two tasks are allocated to the exit side, $P_h^*=\phi$ and hence $P_i^*=\phi$ as well. Clearly, the tasks in S_m also have no predecessors and X is dominated by Z using NSPR.

Suppose that a set of sub-problems are dominated by others using MEJR and NSPR. There must be some sub-problems X, pruned by MEJR but not NSPR, and some sub-problems Y, which dominate X. The compatibility is violated when the sub-problems to be pruned only by one dominance rule, and both MEJR and NSPR take effect independently. In other words, the compatibility is violated when there is a pair of X and Y where X is pruned only by MEJR and Y is pruned only by NSPR. As implied by Lemma 3, this situation cannot happen and thus the MEJR and NSPR are compatible for UALBP.

Appendix B. Detailed results

The detailed results obtained by the tested algorithms are publicly available at http://ikucukkoc.baun.edu.tr/datasets/CAIE_BBR_Results.zip.

References

[1] Scholl A, Becker C. State-of-the-art exact and heuristic solution procedures for simple assembly line balancing. European Journal of Operational Research. 2006;168(3):666-93.

[2] Battaïa O, Dolgui A. A taxonomy of line balancing problems and their solution approaches. International Journal of Production Economics. 2013;142(2):259-77.

[3] Miltenburg GJ, Wijngaard J. The U-Line Line Balancing Problem. Management Science. 1994;40(10):1378-88.

[4] Li Z, Kucukkoc I, Nilakantan JM. Comprehensive review and evaluation of heuristics and meta-heuristics for two-sided assembly line balancing problem. Computers & Operations Research. 2017;84:146-61.

[5] Hoffmann TR. Eureka: A Hybrid System for Assembly Line Balancing. Management Science. 1992;38(1):39-47.

[6] Johnson RV. Optimally Balancing Large Assembly Lines with "Fable". Management Science. 1988;34(2):240-53.

[7] Nourie FJ, Venta ER. Finding optimal line balances with OptPack. Operations Research Letters. 1991;10(3):165-71.

[8] Scholl A, Klein R. SALOME: A Bidirectional Branch-and-Bound Procedure for Assembly Line Balancing. INFORMS Journal on Computing. 1997;9(4):319-34.

[9] Scholl A, Klein R. Balancing assembly lines effectively – A computational comparison. European Journal of Operational Research. 1999;114(1):50-8.

[10] Bautista J, Pereira J. A dynamic programming based heuristic for the assembly line balancing problem. European Journal of Operational Research. 2009;194(3):787-94.

[11] Sewell EC, Jacobson SH. A Branch, Bound, and Remember Algorithm for the Simple Assembly

Line Balancing Problem. INFORMS Journal on Computing. 2012;24(3):433-42.

[12] Morrison DR, Sewell EC, Jacobson SH. An application of the branch, bound, and remember algorithm to a new simple assembly line balancing dataset. European Journal of Operational Research. 2014;236(2):403-9.

[13] Fleszar K, Hindi KS. An enumerative heuristic and reduction methods for the assembly line balancing problem. European Journal of Operational Research. 2003;145(3):606-20.

[14] Hoffmann TR. Assembly Line Balancing with a Precedence Matrix. Management Science. 1963;9(4):551-62.

[15] Blum C. Beam-ACO for Simple Assembly Line Balancing. INFORMS Journal on Computing. 2008;20(4):618-27.

[16] Scholl A, Klein R. ULINO: Optimally balancing U-shaped JIT assembly lines. International Journal of Production Research. 1999;37(4):721-36.

[17] Yolmeh A, Salehi N. A branch, price and remember algorithm for the U shaped assembly line balancing problem. arXiv preprint arXiv:170804127. 2017.

[18] Erel E, Sabuncuoglu I, Aksu BA. Balancing of U-type assembly systems using simulated annealing. International Journal of Production Research. 2001;39(13):3003-15.

[19] Sabuncuoglu I, Erel E, Alp A. Ant colony optimization for the single model U-type assembly line balancing problem. International Journal of Production Economics. 2009;120(2):287-300.

[20] Baykasoğlu A, Dereli T. Simple and U-type Assembly Line Balancing by Using an Ant Colony Based Algorithm. Mathematical and Computational Applications. 2009;14(1):1.

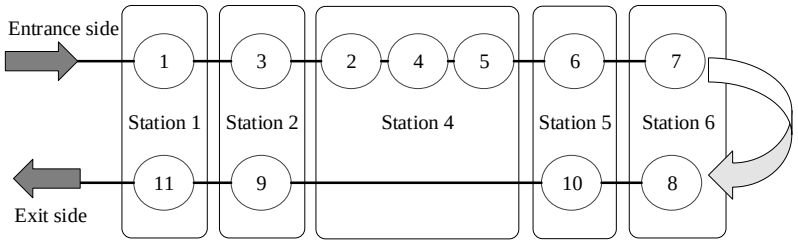
[21] Li Z, Kucukkoc I, Tang Q. New MILP model and station-oriented ant colony optimization algorithm for balancing U-type assembly lines. Computers & Industrial Engineering. 2017;112:107-21.

[22] Otto A, Otto C. How to design effective priority rules: Example of simple assembly line balancing. Computers & Industrial Engineering. 2014;69:43-52.

Highlights

1. Branch, bound and remember algorithm is improved for UALBP.
2. Two new dominance rules are developed and proved.
3. A comprehensive study is carried out to test eight BBR methods.
4. BBR methods outperform two current best exact methods, ULINO and BPR.
5. New optimal solutions and upper bounds are achieved for tested benchmarks.

Graphical Abstract



Algorithm 1: Branch, bound and remember algorithm

% Phase I

Step 1: Achieve UB by modified Hoffman heuristic.

% End of Phase I

Step 2: Obtain global lower bound at the root using $\mathrm{LB}_{root} = max(LB1, LB2, LB3, BPLB)$.

Step 3: If $\mathrm{UB} = \mathrm{LB}_{root}$, terminate; otherwise, go to Step 4.

% Phase II

Step 4: Run cyclic best-first search strategy and update UB when smaller UB is achieved. If the termination criterion is satisfied, terminate.

% End of Phase II

Step 5: If $\mathrm{UB} = \mathrm{LB}_{root}$ or termination criterion is satisfied, terminate; otherwise, go to Step 6.

% Phase III

Step 6: Run breadth-first search strategy and update UB when smaller UB is achieved. If $\mathrm{UB} = \mathrm{LB}_{root}$ or termination criterion is satisfied, terminate this procedure.

% End of Phase III