

Received May 31, 2020, accepted July 17, 2020, date of publication August 4, 2020, date of current version August 17, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3014269

A Three-Stage Optimization Method for Assembly Line Balancing Problem

QIDONG YIN^{1,2}  AND XIAOCHUAN LUO^{1,2,3} , (Member, IEEE)

¹College of Information Science and Engineering, Northeastern University, Shenyang 110819, China

²State Key Laboratory of Synthetical Automation for Process Industries, Northeastern University, Shenyang 110819, China

³State Key Laboratory of Rolling and Automation, Northeastern University, Shenyang 110819, China

Corresponding author: Xiaochuan Luo (luoxch@mail.neu.edu.cn)

This work was supported in part by the National Key Research and Development Program of China under Grant 2017YFB0304100 and Grant 2019YFB1705002, in part by the National Natural Science Foundation of China under Grant 51634002, and in part by the Open Research Fund from the State Key Laboratory of Rolling and Automation, Northeastern University, under Grant 2018RALKFKT008.


ABSTRACT Considering the characteristics of multimodels production pattern on assembly line, the assembly line balancing problem which is non-deterministic polynomial hard becomes more challenging to complete. In this article, we propose a reformulation of simple assembly line balancing problem based on Dantzig-Wolfe decomposition. New models of the master problem and subproblems in this algorithm are built. We implement a branching rule which is suited to seeking integer solutions of the problem. A new three-stage branch-and-price algorithm is designed to accelerate the process of searching the branch-and-bound tree. Extensive computational experiments on benchmark data sets, as well as a real industry case are conducted. The numerical results validate the feasibility and effectiveness of the proposed method which performs efficiently on various cases. Effects on optimization results considering the characteristics of the instance sets are analyzed. Results show that the three-stage branch-and-price algorithm is superior to the classic branch-and-price algorithm in terms of solution quality and computing time.

INDEX TERMS Assembly process planning, line balancing, branch and price algorithm, column generation.

I. INTRODUCTION

The production system of passenger vehicle manufacturing consists of four main areas: Press shop, Body shop, Paint shop and Assembly shop. Assembly represents the final phase in which parts are sequentially installed on the semi-finished vehicle body as it moves from one work station to the next. Fig.1 shows the structure of main assembly line which consists of more than 300 work stations (gray squares with green borders) at Plant Tiexi, BMW Brilliance. Six models are produced in Plant Tiexi currently, and there are 2000+ parts designed to be installed on every vehicle. Planners are responsible for designing a line balancing scheme for each model of cars according to the structure of the assembly line, the production rate and work force. However, the line balancing work is done manually depending on planners' own experiences now. These plans need to be updated often as orders change and new models are planned to be produced. Thus, the line balancing work becomes more challenging to be finished. It is crucial to design an algorithm which

could solve the problem efficiently and correctly. The problem of assigning tasks to sequential stations in such a way that one or more objectives are optimized subject to some specific constraints is called the assembly line balancing problem (ALBP). The basic formulation of assembly line balancing problem, in which one worker is assigned to a work station, is known as the simple assembly line balancing problem (SALBP). SALBP can be classified into two groups: simple assembly line balancing problem type-1 (SALBP-I) and simple assembly line balancing problem type-2 (SALBP-II). In SALBP-I, the objective is to minimize the number of work stations for a given cycle time, while in SALBP-II the objective is to minimize the cycle time for a given number of work stations. SALBP-I could be applied at the stage of planning assembly process for a new vehicle before its start of production (SOP) or optimization of production cost after SOP. SALBP-II matches for the reconfiguration of the assembly line when the production rate increase or the take rate of specific car model changes during production. We study SALBP-I to deal with practical demands of automotive industry that more models are planned to be produced on one assembly line.

The associate editor coordinating the review of this manuscript and approving it for publication was Shaoyong Zheng .

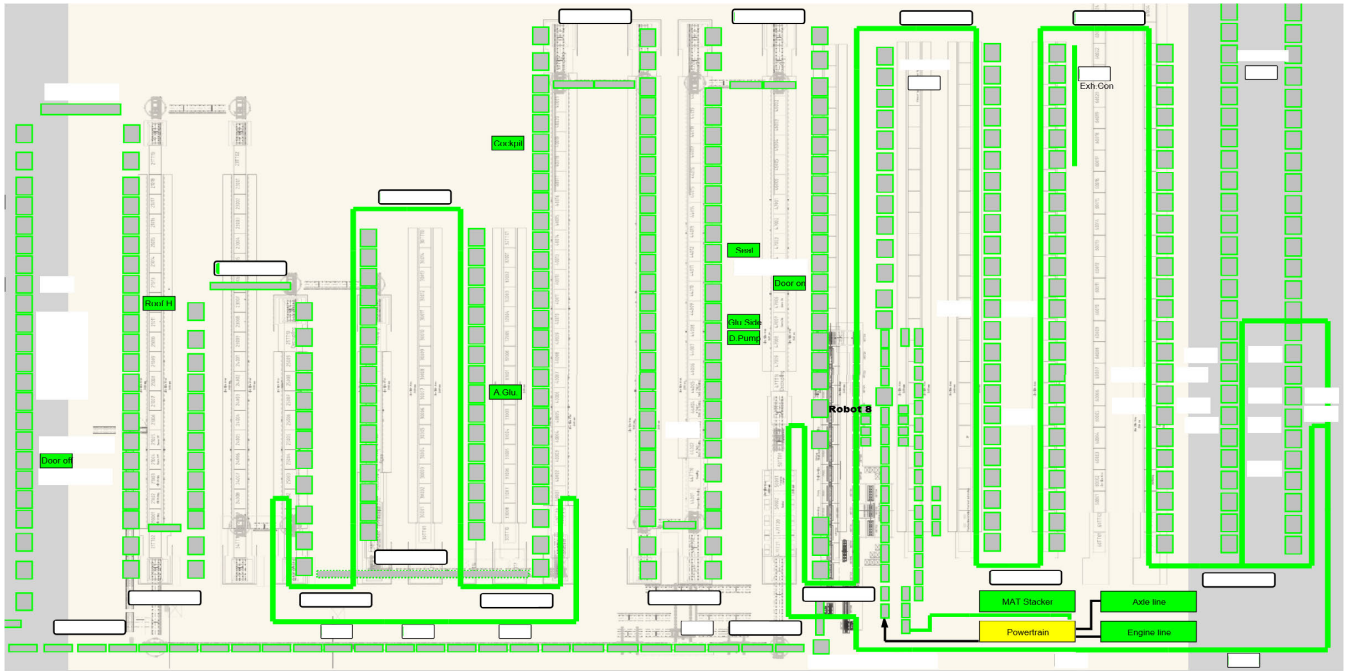


FIGURE 1. Assembly line system at Plant Tiexi, BMW Brilliance Automotive.

The remainder of this article is structured as follows: Section 2 discusses research works of SALBP-I done in literature and the column generation method used in scheduling and planning area. In Section 3, we describe the objective and constraints of SALBP-I, and present a new reformulation of SALBP-I based on the Dantzig-Wolfe decomposition method. In Section 4, we develop a branch-and-price algorithm which integrates the column generation method in the frame of branch-and-bound approach to solve the mathematical model built in Section 3. In this article, a specific branching strategy is implemented in the branch-and-price algorithm. In Section 5, experiments based on benchmark data sets as well as a test for the real production case, which locates in the production area of the vehicle's frontend pre-assembly, at one premier brand OEM's assembly shop are conducted. Computational results are presented and analyzed Section 6. Conclusions and further research work directions are stated in the final part.

II. LITERATURE REVIEW

There exist a large number of models and methodologies for the assembly line balancing problem, and we will focus our review on the researches of the simple assembly line balancing problem.

Baybars [1] presented a survey of exact algorithms for the simple assembly line balancing problem. He concluded that only the enumerative techniques (i.e., branch and bound (B&B) or searching methods) had been extensively applied to SALBP-I among the four primary approaches to solve integer programming (IP) problems (cutting plane techniques,

enumerative techniques, partitioning methods and group theory). Scholl and Becker [2] also reviewed the exact procedures for SALBP-I. He classified most exact solutions into two subdivisions: branch and bound (B&B) procedures and dynamic programming (DP) approaches. Charlton and Death [3] defined a general B&B method for machine scheduling. They utilized the solution of a critical path problem to yield a smallest lower bound for each iteration of the algorithm. Jackson [4] first constructed the algorithm for SALBP-I, using an unconventional DP method. Held *et al.* [5] also reported a new DP algorithm with the introduction of the notion of 'feasible subsets' and 'feasible sequences'. Due to the severe computational and storage requirements of the DP methods, few other significant DP approaches were reported until Schrage and Baker [6] proposed an efficient method within the framework of DP approach. Queyranne [7] has also presented a hybrid DP and B&B models, suggesting that such an approach may be more efficient compared to individual DP or B&B approaches. Besides, Bockmayr and Pizaruk [8] developed a branch and cut procedure based on integer programming formulations with additional valid inequalities and constraint programming techniques. Pinnoi and Wilhelm [9], [10] also proposed branch and cut procedures for SALBP-I connected with vertical balancing as well as for more general problems. Recently, Peeters and Degraeve [11] presented a Dantzig-Wolfe type formulation of SALBP-I. Linear programming (LP) relaxation of this formulation was solved by using column generation combined with subgradient optimization. Pastor and Ferrer [12] presented an improved

mathematical program to solve SALBP. They provided a more effective mathematical model by adding an additional set of constraints. An initial pre-process is implemented to calculate the range of work stations for different tasks. Vila and Pereira [13] studied the assembly line worker assignment and balancing problem using a branch, bound and remember algorithm. The branch, bound and remember algorithm utilized the characteristics of a dynamic programming method.

Intelligent algorithms are also widely used on various types of the general assembly line balancing problem. Rubinovitz and Levitin [14], Kim *et al.* [15] and Sabuncuoglu *et al.* [16] developed genetic algorithms for the SALBP. Ponnambalam *et al.* [17] presented a multiobjective genetic algorithm for solving the SALBP with a given cycle time. Yu and Yin [18] designed an adaptive genetic algorithm in which the probability of crossover and that of mutation are dynamically adjusted according to the individual's fitness value. Baykasoglu [19] proposed a simulated annealing algorithm for simple assembly line balancing problems. Seyed-Alagheband *et al.* [20] constructed a mathematical model and a novel simulated annealing (SA) algorithm to solve the general assembly line balancing problem aiming at minimization of the cycle time with a given number of work stations. Lapierre *et al.* [21] stated a new tabu search algorithm (TSA) for SALBP-I and verified it with a real industrial data set. Annarongsri and Limnararat [22] presented a hybrid tabu search method, which combined the tabu search with the genetic algorithm, for the SALBP-I. Fattahi *et al.* [23] developed a mathematical model and ant colony optimization method for SALBP-I. Lai and Liu [24] applied an ant colony algorithm to solve the SALBP-II.

Due to the fact that SALBP is a class of NP-hard optimization problem, effective exact and heuristic procedures can only obtain feasible solutions on medium size instances. Both the number of nodes generated in branch-and-bound procedures and the computational demands of a DP method grow exponentially as the problem size increases [1]. Most exact algorithms have large computer memory requirements, since the enumerating procedures which assign tasks to stations do not strictly work stage-by-stage [2]. In general, when the number of tasks is large, all exact algorithms fail, in the sense that the CPU times grow very rapidly [1]. Thus, algorithms of SALBP for solving large-scale instances are necessary to be studied.

Column generation has proven to be one of the most successful approaches for solving large-scale integer programming (IP). Three types of column generation approaches have been used to solve problems with a huge number of columns. Rather than enumerating so many columns explicitly, these methods deal with them implicitly, generating a selected set. The master problem, which contains a part of columns of original problem's solution, is defined as the Restricted Master Problem (RMP) in column generation.

In the Type I approach, an auxiliary model is employed to generate the set of feasible columns, and a RMP is built to optimize over those generated columns obtaining the best subset. Besides accepting columns from the auxiliary model, there is no interaction between the restricted master problem and the auxiliary model. This method was successfully applied to airline crew scheduling problems in the 1960s and 1970s [25], [26]. Type II approach of column generation includes the interaction between the restricted master problem and subproblems. The restricted master problem provides dual variable values as the coefficients of the subproblems' objective function. These data could direct the search for improving columns in subproblems. The subproblem chooses nonbasic columns which could improve the master problem with the best reduced cost. Gilmore and Gomory [27] first devised the Type II methodology to the cutting stock (CS) problem. A comprehensive algorithm was designed to form a prototype for other applications. Gilmore and Gomory [28] also proposed further problem-specific techniques to facilitate this algorithm. Balinski and Quandt [29] first suggested formulating the vehicle routing problems with time windows (VRPTW) as a set covering problem and Cullen *et al.* [30] first devised a heuristic that exploited the related set partitioning form. Wilhelm [31] applied the column generation method to the assembly system design with tool changes problem. Type III column generation applied Dantzig-Wolfe decomposition [32] to the linear relaxation of an IP. The restricted master problem provides dual values to the subproblem which generates the improving column during the iteration. Appelgren [33] first used the Dantzig-Wolfe decomposition to solve the ship scheduling problem which is an IP. Appelgren [34] integrated the Dantzig-Wolfe decomposition into a branch and bound searching frame dealing with the fractional solution. Type III column generation are successfully applied in cutting stock problems. Vance *et al.* [35], [36] proposed a branch and price algorithm to solve the CS problem. Savelsbergh [37] presented a method of branch-and-price for the generalized assignment problem. However, few works are done with applying the branch-and-price method to SALBP-I so far.

The aim of this article is to develop a branch-and-price algorithm to solve SALBP-I. The scale of practical line balancing problem in automotive industry becomes larger than decades ago, and we analyze the feasibility and efficiency of the proposed algorithm. We reformulate a new model for Dantzig-Wolfe decomposition of the SALBP-I that maintains precedence constraints in the master problem. Thus the subproblem could be designed as a knapsack problem with side constraints which is solvable in pseudo-polynomial time [38]. In our study, we construct an efficient frame of a three-stage branch-and-price algorithm, in which a heuristic algorithm gives the initial solution in the 1st stage, column generation procedures are integrated in the 2nd stage and an IP model is applied to solve the SALBP-I in the 3rd stage, that could

obtain the optimal solution in relatively short time. Effective branching strategy is determined for the unique characteristic of the column generation method.

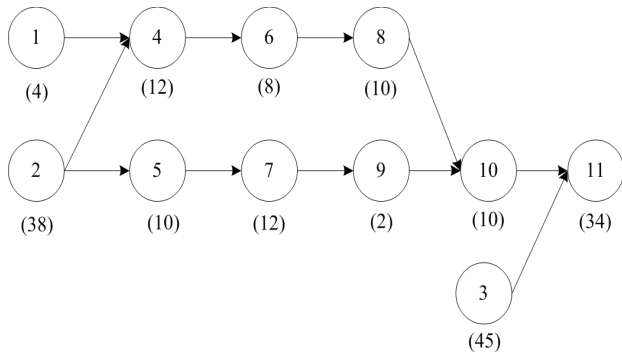


FIGURE 2. Precedence graph of an instance with 11 tasks.

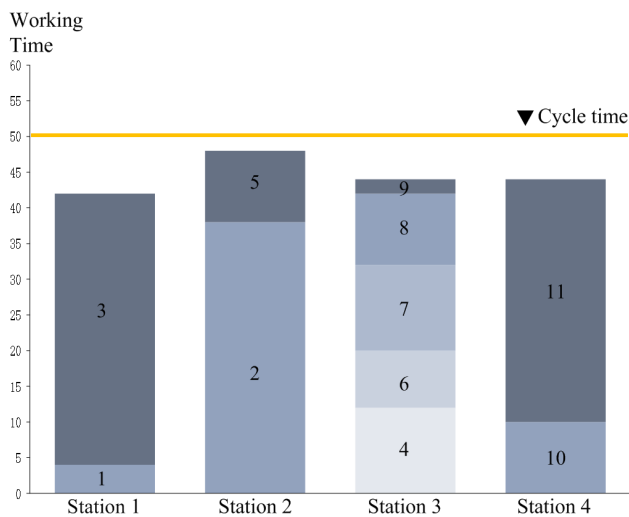


FIGURE 3. Solution example of the instance set with 11 tasks.

III. PROBLEM DESCRIPTION AND FORMULATION

An assembly line is the manufacturing process in which vehicle components are added to an unfinished vehicle body in a sequential manner along a serial flow line. Fig. 2 shows an example of a precedence graph of an instance with 11 tasks. The number in the node represents the task number, and the value in the bracket is operation time for each task. The bar chart in Fig. 3 displays an optimal line balancing solution for this example. Pillars of different colors are marked with the task number 1 to 11, and the height of the pillar matches the length of task’s operation time. The task assignment to the four stations is: Station₁ = {1, 3}, Station₂ = {2, 5}, Station₃ = {4, 6, 7, 8, 9}, and Station₄ = {10, 11}. We illustrate some basic concepts used in the process of assembly production in following texts.

1) *Work station*: A station is the basic unit constructing an assembly line where a number of operations are performed in a fix range of area.

- 2) *Production rate*: The Production rate means the hourly output of finished vehicles on the assembly line associated with the operation or sequence of lines.
- 3) *Cycle time*: The cycle time is planned according to the production rate of an assembly line. One cycle time begins to count when a product arrives at a station, and ends until the following product enters the same station. It gives a limitation that the workload of a worker should be less than the cycle time.
- 4) *Task*: The task is the operation in an assembly process. The sum of all operations adds up to the total work content for a vehicle assembly. The time it takes to perform a task is called operation time.
- 5) *Workload*: The workload means the ratio of sum of one worker’s total operation times needed for a vehicle to the station’s cycle time.
- 6) *Precedence relationship*: Precedence relationship defines the order in which operations of tasks should follow. Precedence constrains can be summarized in a precedence graph (activity-on-arrow diagram) that shows which operation has to be completed before a specific operation can start.
- 7) *Predecessor*: Predecessor refers to the set of tasks that must be done previous to one task according to the precedence relationship.
- 8) *Successor*: Successor refers to the set of tasks that follow one task in the precedence relationship.

A. PROBLEM STATEMENT

Based on the above explanations of special terms, we define the SALBP-I as: The purpose of SALBP-I is to obtain the optimal line balancing plan with a minimum number of stations under a given cycle time. The line balancing solution must include the assignment of all tasks in the data set.

To ensure the feasibility of line balancing solution, we seek out rules that formed in the planning work. We state the key instructions for the assembly line balancing in details below.

1) ASSEMBLY SEQUENCE

Assembly sequence is defined as the order in which parts are installed on the vehicle. In the study of line balancing problem, the precedence relations of tasks are edited on the basis of assembly sequence. Line balancing solution should satisfy the restrictions of the precedence relations. Since the assembly sequence depends on the structure of product, some parts must be installed first before its successors. Mechanical interference between parts happens if the operation breaks the precedence restrictions which may results in a shutdown of the assembly line.

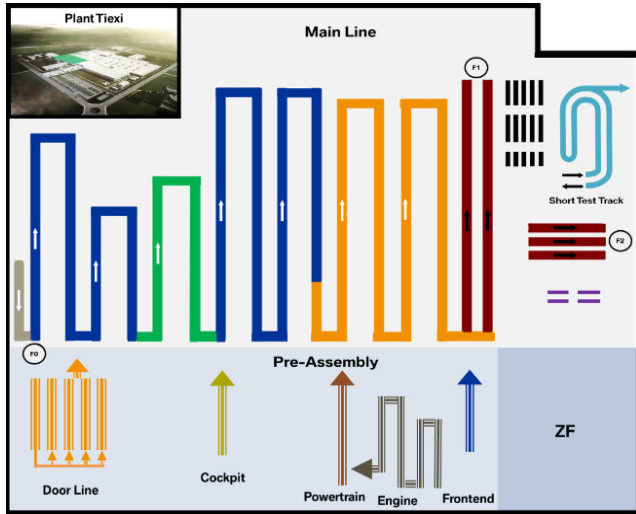


FIGURE 4. Structure of the automotive assembly shop.

2) ALLOWED WORKPLACES

The assembly line is usually designed as a main assembly line connected with several pre-assembly lines in the assembly shop. Furthermore, the main assembly line can be partitioned into several areas with different types of the vehicle’s conveyor which are marked with different colors in the structure layout in Fig. 4. Different structures of the conveyers and hangers make it possible to assemble parts to specific place on the vehicle for the process and ergonomic reasons. Thus, some operations should be completed on the recommended area. For example, all the operations of installing parts underneath the vehicle body are distributed to the line with tilting hangers (green area in Fig. 4: the tilting line) which could lift the car vertically and adjust the tilting angle. Parts assembly of doors, cockpit and frontend are completed in the independent pre-assembly areas noted in the layout. As a result, the line balancing work in the industry practice is done independently according to the assembly line’s different areas.

3) RELATIONS BETWEEN TASKS

Some tasks require to be done continuously without a break. Thus we create a group for this kind of tasks. Considering special process requirements, some tasks must be completed on the same station, while some operations are exclusive in one station that there must be the station gap between them. For those parts with a large size, they need two or more workers cooperate together to finish the installations. These tasks are marked as simultaneous work.

4) GUIDELINES OF THE WORK ORGANIZATION

All of operations are finally edited in the Standard Work Specifications (SWS) document. The list of SWS states a clear description of the tasks content and how to finish it on each station. Total working time for one worker should be less than the cycle time. Besides, the worker is not allowed

to take two tasks at the same time. One task description usually includes several physical actions. The calculation of workload of the worker is based on these actions. Using the standard methods of time measurement planners could compute the accurate task time.

TABLE 1. Definitions of the set symbols.

Set	Definition
I	The set of tasks
J	The set of stations
T	The set of feasible assignments
P_i	The set of immediate predecessors of task i
P_i^*	The set of total tasks which precede task i
F_i	The set of immediate successors of task i
F_i^*	The set of total tasks which follow task i
A	The set of the assignment matrix
Q	The set of columns of the assignment matrix
L	The set of task pairs in the left branch
R	The set of task pairs in the right branch

TABLE 2. Definitions of the parameter symbols.

Parameter	Definition
ct	Cycle time
t_i	the operation time of task i
π	the set of dual variables for constraint set (14)
μ	the set of dual variables for constraint set (15)
m_k^j	Assignment pattern k of station j
rc	the reduced cost

B. NATURAL FORMULATION OF SALBP-I

The definitions of the sets, parameters, and decision variables used later in this article are presented in Table 1, Table 2 and Table 3, respectively. We describe the SALBP-I as an IP formulation:

$$\text{Minimize } \sum_{j \in J} y_j. \quad (1)$$

Subject to

$$\sum_{j \in J} a_{ij} = 1 \quad \forall i \in I \quad (2)$$

$$\sum_{i \in I} t_i \cdot a_{ij} \leq ct \cdot y_j \quad \forall j \in J \quad (3)$$

$$\sum_{j \in J} j \cdot a_{hj} \leq \sum_{j \in J} j \cdot a_{ij} \quad \forall i \in I, h \in P(i) \quad (4)$$

$$a_{ij} \in \{0, 1\} \quad \forall i \in I, j \in J \quad (5)$$

$$y_j \in \{0, 1\} \quad \forall j \in J \quad (6)$$

The objective function of the SALBP-I is to complete all tasks assignment with the minimum number of stations. Constraints set (2) ensure that all tasks in the data set are done and every task (i) in the set is assigned to one work station. Constraints set (3) define the cycle time constraint for work stations. The sum of operation time of task on one station

TABLE 3. Definitions of the variables.

Variables	Definition
a_{ij}	If task i is assigned to station j ,=1;otherwise,=0
y_j	Indicator variable, if station j is open, =1;otherwise,=0
λ_k^j	If pattern k of station j is used in the solution,=1; otherwise,=0
z	Decision variable of the one station's assignment
x_i	If task i is selected in the new pattern, =1; otherwise,=0

for each vehicle should be less than the cycle time of the assembly line. Once a station has been assigned to a task, then this station will be noted as open in the solution and the value of variable $y_j = 1$. Constraints set (4) describe the sequence requirement for all tasks. Any preceding operation for one specific task should not be assigned to stations later than the station where its reference task locates. The predecessor's assignment is acceptable either before or in the same station with its reference task.

C. DANTZIG-WOLFE REFORMULATION

Dantzig-Wolfe decomposition is a standard way to decompose an integer programming model into a linear programming master problem and one or several subproblems. Dantzig-Wolfe decomposition has a close connection to column generation method. Instead of focusing on which station a particular task is assigned to, the possible plans used to design the solution for each station are regarded as variables in the new formulation.

We construct a reformulation of SALBP-I:

$$\text{Minimize } \sum_{j \in J} y_j. \tag{7}$$

Subject to

$$\sum_{j \in J} \sum_{k \in T_j} \lambda_k^j m_{ik}^j = 1, \quad \forall i \in I \tag{8}$$

$$\sum_{j \in J} j \cdot \sum_{k \in T_j} \lambda_k^j m_{hk}^j \leq \sum_{j \in J} j \cdot \sum_{k \in T_j} \lambda_k^j m_{ik}^j \quad \forall i \in I, h \in P(i) \tag{9}$$

$$\sum_{k \in T_j} \sum_{i \in I} t_i \lambda_k^j m_{ik}^j \leq ct \cdot y_j \quad \forall j \in J \tag{10}$$

$$\lambda_k^j \in \{0, 1\} \quad \forall k \in T_j, j \in J \tag{11}$$

$$y_j \in \{0, 1\} \quad \forall j \in J \tag{12}$$

The objective of the reformulation is still to calculate the minimum number of work stations same as the original model (1). Constraint set (8) is derived from assignment constraint set (2). Sequence constraints of set (4) are reformulated in constraint set (9). Constraint (10) determines the cycle time constraint. Also, the value of variable $y_j = 1$ if the feasible pattern k for station j is kept in the solution.

However, there will be an extremely huge number of columns in the matrix m when the size of task set grows larger. It is not feasible to enumerate all of the possible combinations

of task assignment. We implement the column generation method to solve the reformulated model of SALBP-I in another section below. This method transforms a problem into a master problem and pricing problems for improving the tractability of large-scale problems.

D. COLUMN GENERATION METHOD

The fact that, there are a huge number of these combinations between the tasks and stations, results in considerable variables in the original model of SALBP-I. In the column generation method, the line balancing problem is designed as two parts: the restricted master problem and the subproblem which is also called the pricing problem. The restricted master problem covers partial sets of the assignment patterns. Other solution patterns are generated through solving the subproblem.

We derive the master problem based on the D-W decomposition of the original problem model as follows.

$$\text{Minimize } \sum_{j \in J} \sum_{p \in Q} z_j^p \tag{13}$$

Subject to

$$\sum_{j \in J} \sum_{p \in Q} A_{ij}^p z_j^p = 1 \quad \forall i \in I \tag{14}$$

$$\sum_{j \in J} \sum_{p \in Q} j \cdot A_{hj}^p z_j^p \leq \sum_{j \in J} \sum_{p \in Q} j \cdot A_{ij}^p z_j^p \quad \forall i \in I, h \in P(i) \tag{15}$$

$$0 \leq z_j^p \leq 1 \tag{16}$$

The minimization of the station number forms as the objective function value in the master problem. The matrix A consists of the current set of task distributions. Each column represents an assignment pattern of one station. A heuristic approach for solving SALBP-I is developed to obtain the initial matrix. Details will be introduced in next section.

Task assignment constraints (8) and assembly sequence constraints (9) in the reformulation of model are kept in the restricted master problem. Constraints set (14) that matches the constraints set (8) in the original formulation ensures the task assignment constraints. Constraints set (15) makes sure that original sequence constraints set (9) could still be satisfied after the reformulation.

Variables in (16) which are relaxed binary variables of (11) define the master problem as a linear programming. The restricted master problem considers only a subset of the columns. Additional columns can be generated for the RMP by solving a subproblem. To check optimality, the subproblem, also called the pricing problem, is solved to identify columns to enter the basis of the master problem's parameter set A . Then the linear programming problem is reoptimized after adding this kind of columns.

According to the theory of the simplex algorithm, a column with the negative reduced cost value can improve the current solution. Thus, we set the minimum reduced cost as objective value of the pricing problem, and the constraints inherit

from constraints of the original problem. The most negative reduced cost rc of a column j in the master problem is defined as:

$$rc = \text{Min} \left(1 - \sum_{i=1}^n \pi_i x_i \right) = 1 - \text{Max} \sum_{i=1}^n \pi_i x_i \quad (17)$$

where π represents the dual cost vector consists of the dual variables of the restricted master problem.

We formulate the subproblem as a 0-1 knapsack problem with side constraints.

$$\text{Max} \left(\sum_{i=1}^n \pi_i x_i + \sum_{i \in P0} \sum_{h \in \text{Pre}(i)} j \cdot \mu_i^h \cdot (x_i - x_h) \right) \quad (18)$$

Subject to

$$\sum_{i \in I} t_i x_i \leq ct \quad (19)$$

$$x_{i1} + x_{i3} - x_{i2} \leq 1 \quad \forall i1 \in P(i2), i3 \in F^*(i2) \quad (20)$$

$$x_i \in \{0, 1\} \quad \forall i \in I \quad (21)$$

The objective function is equal to searching for the minimization of the reduced cost for the master problem. The coefficients of variables in (18) are related to the dual variable values associated with constraints set (14) and (15). Parts of constraints of the knapsack problem are the same as the cycle time constraints of the original line balancing problem, namely the constraint set of (3). The cycle time value of the original problem gives a weight limit in the knapsack problem model. Another set of constraints is added to the knapsack problem: task sequence constraints (20). These constraints guarantee that solutions could satisfy the sequence constraints set (4).

The solution of the pricing problem represents one possible pattern of task assignment. The new pattern can be added into the master problem's solution pool when its reduced cost is negative. For each station, a specific pricing problem is constructed to calculate the minimum reduced cost. Pricing problems are designed to be solved in the order of the station's sequence. After getting an improved column, we add it into the matrix A of the master problem.

IV. THREE-STAGE BRANCH-AND-PRICE ALGORITHM

A. OVERVIEW OF ALGORITHM

The three-stage branch-and-price algorithm consists of three parts of computation. The flow chart of the algorithm is illustrated in Fig. 5. In the first stage, an initial solution for the instance of SALBP-I is given applying a heuristic approach. The object value of the initial solution acts as the upper bound Z_{best} in the algorithm. The lower bound in our algorithm is calculated as the simple lower bound $LB_1 = \lceil \text{sum}(t)/ct \rceil$. The initial solution also constructs the root node of the branch and bound tree. Details of this initial algorithm are described in the next subsection B .

In the second stage, we implement the iteration procedure of column generation. At children nodes in the searching tree, the LP relaxation model of the line balancing problem is

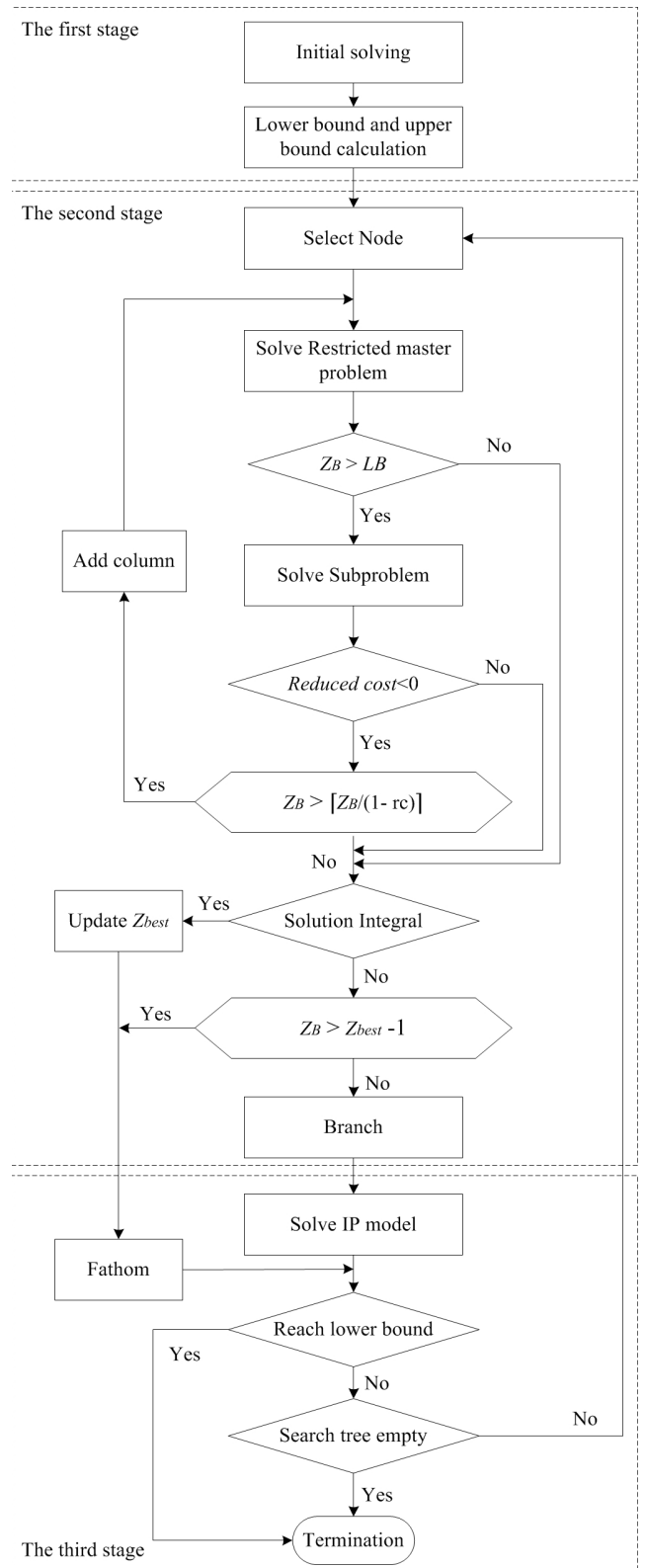


FIGURE 5. Three-stage branch-and-price algorithm.

solved using the column generation method. We define this problem as “restricted master problem” because the initial reformulation of master problem only includes a subset of

the solution. Other columns with negative reduced cost are generated iteratively by solving the pricing problems. Given the optimal dual solutions which act as the multipliers in the subproblem's objective function, we use *Gurobi* optimizer to solve subproblems. To identify a new column, the negative reduced cost requirement of the master problem needs to be satisfied by checking the optimal value of subproblem. We add the optimal solution into the master problem as a new column of the parameter set and repeat this iteration until the negative reduced cost no more exists.

Iterations between the master problem and the pricing problem stop when three cases happen: The first condition works if the basic solution value of the master problem reaches the lower bound $Z_B \leq LB$. Secondly, the iteration breaks when the reduced cost got from the new pricing problem is nonnegative $rc \geq 0$ which means that current solution of the master problem cannot be improved any more. The third condition interrupts the iteration when the value of the reduced cost rc is small enough. Since the number of the work station is integer, the final result should not be smaller than the rounded up value of the basic solution value. Thus the final result will not be smaller when the reduced cost locates within a certain range $Z_B = \lceil Z_B / (1 - rc) \rceil$.

When the iteration stops, we analyze the possible result obtained at the current phase. Options should be taken under different conditions:

- 1) Whenever an integer solution in node h is found, if $Z_h < Z_{best}$, then solution of node h becomes the new incumbent solution, and the node h is pruned after updating Z_{best} .
- 2) For fractional node i which holds $Z_i > Z_{best} - 1$, then node i is fathomed for bounding.
- 3) The node with no feasible solution should also be fathomed.
- 4) Branching is performed on fractional node i with a solution that $Z_i \leq Z_{best} - 1$.

If we cannot obtain an integer solution at the second stage, branching takes place and it generates two new children nodes in the branch-and-bound tree. In the third stage of the branch-and-price algorithm, an IP model of the Dantzig-Wolfe reformulation (7)-(12) is applied to solve SALBP-I. The parameters of the IP model are based on the columns generated in the second stage. This procedure is designed to seek integer solutions. If we achieve the optimal solution, the algorithm terminates and the final solution is updated, otherwise calculation on active nodes continues. The list of active nodes in the searching tree consists of nodes that are either not solved or solved but with fractional solutions. When no active node left in the branch-and-bound tree the algorithm terminates. The current incumbent solution becomes the final solution.

The lower and upper bounds used in the branch-and-price algorithm could improve the efficiency of tree searching. A good lower bound of optimal IP solution value decreases the iteration times of column generation as the first stopping

criteria stated above. The lower bound is set to the minimum value of work station over all active nodes in this algorithm. And the upper bound value is updated according to the value of $\lceil Z_B \rceil$ when the column generation process finishes in the 2nd stage and the value of IP solution in the 3rd stage. The upper bound is the current optimal integer solution value. Nodes with the value exceeds the upper bound could be pruned without further branching. If an integer solution is obtained, this is the best feasible integer solution that can be obtained in this part of the tree, thus no more branching is needed on this node [33]. Z_{best} can be updated to reduce the gap between the lower bound and the upper bound.

B. INITIAL SOLUTIONS

We need to determine the initial restricted master problem to start the branch-and-price algorithm. The optimal values of dual variables can be obtained through solving the initial restricted master problem. Based on these values we devise the object function of subpricing problems. Not only do we need to set an initial restricted master problem, but also at every branching node in the searching tree of the branch-and-price algorithm, an available restricted master problem has to be initialized to kick off the iteration process of column generation. A heuristic algorithm is designed to construct the initial solution of the restricted master problem in this work. The pseudo-code is described below.

Heuristic Algorithm 1 Pseudo-Code of Initial Solution

```

1: while the Pool not empty do
2:   if  $t_i >$  the time capacity of active station  $j$  then
3:     Set a new station  $j + 1$  active
4:   end if
5:   for task in Pool do
6:     if  $L = \emptyset$  and  $R = \emptyset$  then
7:       assign the current task to active station
8:       update the pool set
9:     end if
10:    if  $L = \emptyset$  and  $R \neq \emptyset$  then
11:      exclusive tasks checking
12:    end if
13:    if  $L \neq \emptyset$  then
14:      search tasks for packing
15:      assign the packing tasks to active station
16:      update the pool set
17:    end if
18:  end for
19: end while

```

Since the branching is realized through adding constraints in the pricing problems, the initial restricted master problem should not be violent against these constraints. As the searching trees grow, the number of task pairs that should be combined or disjoint will increase in deeper layers. Thus, we need to construct the specific initial master problem which has a feasible LP relaxation at different nodes. The methodology of the heuristic algorithm for the initialization is based on the

task oriented approach. Tasks in the pool, whose preceding operations are totally assigned already, will be updated during the process of solution searching. Tasks are classified into three categories after checking their feasibility of fixing to current station: Firstly, task without any related constraint pair can be assigned directly according to the current station's time capacity. Secondly, tasks, which are included in the disjointing pair, should satisfy exclusive constraints that their exclusive tasks are not in current station. Thirdly, we must pack tasks that appear in one joining pair constraint together through grouping them as one task in later computing process. Tasks which belong to the set of middle range of the joining pair in the precedence relationships need to be packed simultaneously. New station will be created when the capacity of current station is not enough to accept any unassigned task in the pool. Searching for a feasible solution continues until the pool set is empty which means that every task has a corresponding assignment.

C. SEARCH STRATEGY

The search strategy controls how to select unexplored nodes in the branch-and-bound tree. Search strategies affect the computer memory requirements and computation time of the branch-and-price algorithm. Different search strategies have specific characteristics which are suitable to variant problems. Depth-first search, breadth-first search and best-first search are three common search strategies widely used in the branch-and-bound algorithm.

Depth-first search (DFS) strategy navigates the search path in the order of relation of the current node. At current node in the branch-and-bound tree, new branching nodes generated from this node will be explored first for further study. This is realized through setting a stack to store the list of unexplored nodes. We select the top item in the stack for exploration, and then remove it after obtaining a feasible solution. Children nodes based on the branching procedure of this fractional solution are inserted on the top of that stack. Thus, in the depth-first search tree, the node which is created most recently will be calculated first.

Breadth-first search deals with the list of branch-and-bound nodes in the data structure of a queue. Contrary to the depth-first search method, breadth-first search manage the unexplored nodes in a first-in, first-out sequence. Breadth-first search runs well on unbalanced search trees since this strategy could always detect the optimal solution that is closest to the root of the tree.

Best-first search compare the values of lower bound of active nodes and select the node with smallest value as the next branching node. Best-first search is conducted by managing the active nodes list in a data structure of heap and setting the lower bound value as the key. As a result, best-first search often discover good solutions earlier in the search tree. The last strategy tested in our algorithm is called cyclic best-first search. This approach, which is originally named distributed best-first search, is a hybrid method between depth-first search and best-first search. Unexplored nodes list

is divided into a group of heaps according to the depth of node in the searching tree. Cyclic best-first search picks out the smallest node from each heap to branch and takes the same operation at the next heap until all heaps are empty. Cyclic best-first search will explore the node with the best value at depth 0, then depth 1, and so on; upon reaching the deepest layer of the search tree, it will repeat the process starting from depth 0. The choice of search strategy has potentially significant consequences for the amount of computation time required for the branch-and-price procedure, as well as the amount of memory used.

V. IMPLEMENTATION DETAILS

A. BRANCHING RULES

The restricted master problem used in the column generation method is a LP relaxation model. The optimal solutions of this model are not necessarily integers when column generation iterations finish at one node. Then the branching works to create new nodes on the searching tree aiming to obtain the integer solution. In the standard branch-and-bound algorithm, branching always acts on fractional variables of the original problem. However this method cannot guarantee obtaining an optimal integer solution in the branch-and-price algorithm. Since the solutions pruned out after branching may appear again in the children nodes' column generation process.

We apply the Ryan-and-Foster branching rule in our branch-and-price algorithm. This branching strategy is based on the following proposition.

Proposition 1: If a basic solution to the master problem is fractional, then there exist two tasks l and m of the master problem such that

$$0 < \sum_{j,p:A_{lj}^p=1, A_{mj}^p=1} z_j^p < 1 \quad (22)$$

Proof: Consider fractional variable z_j^p , Let task l be any task with $A_{lj}^p = 1$. Since z_j^p is fractional, there must exist another basic column with $0 < z_{j'}^{p'} < 1$ and $A_{lj'}^{p'} = 1$. Since there are no duplicate columns in the basis, there must exist a task m such that either $A_{mj}^p = 1$ or $A_{mj'}^{p'} = 1$, but not both. This leads to the following relations:

$$\begin{aligned} 1 &= \sum_{j \in J} \sum_{p \in Q} A_{lj}^p z_j^p \\ &= \sum_{j:A_{lj}^p=1} \sum_{p:A_{mj}^p=1} z_j^p \\ &> \sum_{j,p:A_{lj}^p=1, A_{mj}^p=1} z_j^p \end{aligned} \quad (23)$$

The pair l and m gives the pair of branching constraints.

$$\sum_{j,p:A_{lj}^p=1, A_{mj}^p=1} z_j^p \geq 1 \quad \text{and} \quad \sum_{j,p:A_{lj}^p=1, A_{mj}^p=1} z_j^p \leq 0$$

We follow the structure of binary trees when act branching in the branch-and-bound tree. On the left branch tasks l and

m are required to be covered by the same column. Thus, all feasible columns must have $A_{lj}^p = A_{mj}^p = 1$ or $A_{lj}^p = A_{mj}^p = 0$. While tasks l and m are required to be covered by the different columns on the right branch, and feasible columns must have $A_{lj}^p = A_{mj}^p = 0$ or $A_{lj}^p = 0, A_{mj}^p = 1$ or $A_{lj}^p = 1, A_{mj}^p = 0$. Branching constraints on the left branch equals to combining tasks l and m into a new task. On the right branch, tasks l and m are required to be disjoint. Proposition 1 implies that if no branching pair can be identified, then the solution to the master problem must be integer. The branch-and-price algorithm must terminate after a finite number of branches since there are only a finite number of pairs of tasks.

B. KNAPSACK SUBPROBLEM WITH SIDE CONSTRAINTS

The strategy of branching in this work is preventing the fractional solution from being regenerated in the column generation process. Adding constraints set to the master problem can introduce new dual variables to the pricing problem. Thus, instead of adding the branching constraints to the variables of master problem directly, we guarantee the branching constraints through constructing and solving the pricing problem. Branching constraints can be integrated into the column generation subproblems. On the left branch we have:

$$Max \left(\sum_{i=1}^n \pi_i x_i + \sum_{i \in P0} \sum_{h \in Pre(i)} j \cdot \mu_i^h \cdot (x_i - x_h) \right) \quad (24)$$

$$s.t. \quad \sum_{i \in I} t_i x_i \leq ct \quad (25)$$

$$x_l = x_m \quad \forall l, m \in L \quad (26)$$

$$x_{i1} + x_{i3} - x_{i2} \leq 1 \quad \forall i1 \in P(i2), i3 \in F^*(i2) \quad (27)$$

$$x_i \in \{0, 1\} \quad \forall i \in I \quad (28)$$

The new constraint set (26) requires that the pair of tasks l and m must appear together in new patterns in the left branch. As a result, fractional columns eliminated by the addition of these constraints will not be regenerated again.

Subproblems on the right branch can be formulated as:

$$Max \left(\sum_{i=1}^n \pi_i x_i + \sum_{i \in P0} \sum_{h \in Pre(i)} j \cdot \mu_i^h \cdot (x_i - x_h) \right) \quad (29)$$

$$s.t. \quad \sum_{i \in I} t_i x_i \leq ct \quad (30)$$

$$x_l + x_m \leq 1 \quad \forall l, m \in R \quad (31)$$

$$x_{i1} + x_{i3} - x_{i2} \leq 1 \quad \forall i1 \in P(i2), i3 \in F^*(i2) \quad (32)$$

$$x_i \in \{0, 1\} \quad \forall i \in I \quad (33)$$

On the right branch we add the constraint set (31) to the subproblem. This constraint will prevent columns with tasks assignments $x_i = x_m = 1$ from being generated.

VI. COMPUTATIONAL RESULTS

In this section, we present and analyze experiment results of the proposed algorithm. Benchmark data sets and real production data of the assembly line are employed to test the algorithm. We implement our experiments in Python 3.0 programming environment that constructs an interface with Gurobi 7.5.2 optimizer. The computer on which experiments are conducted has an Intel® Core™ i7-3770 3.40GHz CPU and 8GB RAM memory using the Windows 7 operating system.

TABLE 4. Characteristics of precedence graphs.

Name	n	t_{min}	t_{max}	sum(t_i)	OS(%)	TV
Arcus1	83	233	3691	75707	59.09	15.84
Arcus2	111	10	5689	150399	40.38	568.9
Barthold	148	3	383	5634	25.8	127.67
Bartho12	148	1	83	4234	25.8	83
Bowman	8	3	17	75	75	5.67
Buxey	29	1	25	324	50.74	25
Gunther	35	1	40	483	59.5	40
Hahn	53	40	1775	14026	83.82	44.38
Heskiaoff	28	1	108	1024	22.49	108
Jackson	11	1	7	46	58.18	7
Jaeschke	9	1	6	37	83.33	6
Kilbridge	45	3	55	552	44.55	18.33
Lutz1	32	100	1400	14140	83.47	14
Lutz2	89	1	10	485	77.55	10
Lutz3	89	1	74	1644	77.55	74
Mansoor	11	2	45	185	60	22.5
Mertens	7	1	6	29	52.38	6
Mitchell	21	1	13	105	70.95	13
Mukherje	94	8	171	4208	44.8	21.38
Roszieg	25	1	13	125	71.67	13
Sawyer	30	1	25	324	44.83	25
Scholl	297	5	1386	69655	58.16	277.2
Tonge	70	1	156	3510	59.42	156
Warnecke	58	7	53	1548	59.1	7.57
Wee-Mag	75	2	27	1499	22.67	13.5

A. BENCHMARK DATA SETS

In this part, our experiments use three referenced data sets of SALBP, which is data set of Talbot *et al.* [39], Hoffmann [40] and Scholl [41] respectively. The reported best-known solutions of the instances studied in these data sets provide a benchmark for numerical computations. These three data sets include a total of 25 precedence graphs. Table 4 presents characteristics of these precedence graphs.

The column headings in the Tables of data sets and experiment results mean as follows.

- Name: name of the precedence graph.
- n : the number of tasks.
- ct : cycle time.
- t_{min} : the minimum task time in the instance set.
- t_{max} : the maximum task time in the instance set.
- sum(t): the summation of task time in the instance set.
- OS: the ‘Order Strength’ (OS) is equal to the number of all precedence relations divided by the maximum number of precedence relations.

- TV: the ‘Time Variability Ratio’ is the ratio of maximum task time to the minimum task time.
- UB: upper bound found by the heuristics used to initialize the column generation.
- LB₁: lower bound which equals to $\lceil \text{sum}(t)/ct \rceil$.
- Z*: best-known solution in the literature.
- Nodes: the number of nodes searched in the branch-and-bound tree.
- Iterations: the total times that the column generation procedure runs for finding the final solution.
- CPU: time needed for CPUs to solve the instance of SALBP-I problem.

TABLE 5. Average computing time.

Name	Gurobi (CPU/s)	3-stage (CPU/s)	B&P
Arcus1	200+	160.507	
Arcus2	200+	200+	
Barthold	2.738	200+	
Bartho12	-	-	
Bowman	0.817	0.047	
Buxey	1.28	7.601	
Gunther	0.82	25.5	
Hahn	0.146	0.012	
Heskiaoff	0.209	11.748	
Jackson	1.14	0.089	
Jaeschke	0.718	0.095	
Kilbridge	0.268	9.155	
Lutz1	0.744	2.72	
Lutz2	500+	213.079	
Lutz3	17.817	477.206	
Mansoor	0.087	0.048	
Mertens	0.154	0.051	
Mitchell	0.252	0.574	
Mukherje	5.37	616.042	
Roszieg	0.329	0.004	
Sawyer	30.42	9.425	
Scholl	-	-	
Tonge	500+	290.134	
Warnecke	200+	62.725	
Wee-Mag	80+	73.262	

The average CPU time results of the benchmark data sets are listed in Table 5. The three-stage branch-and-price algorithm outperforms *Gurobi* optimizer for 13 of 25 data sets. *Gurobi* optimizer calculates faster in 9 of 25 data sets. There exist 2 data sets (Barthol2 and Scholl) that the average computing time is too long to obtain the optimal solution for both *Gurobi* optimizer and the three-stage branch-and-price algorithm. We set the limitation of average CPU time as 1000s, computing time beyond this value is noted as (-).

To evaluate the effectiveness of the three-stage branch-and-price algorithm, we analyze that at which phase the optimal solution is obtained in the procedure of optimization. Statistics are listed in Table 6. Totally, the algorithm could found the optimal solution for 48.55% of the instance sets at the 1st stage. And 21.97% of the instances sets got the optimal solution at the 2nd stage which represents the iteration process of column generation. Finally, the added

TABLE 6. Optimal results for different problem characteristics.

Problem Characteristics	Optimal value getting from 3 stages			
	1st stage	2nd stage	3rd stage	
NT	Low	73.81%	11.90%	14.29%
	Middle	34.69%	28.57%	36.73%
	High	43.90%	23.17%	32.93%
OS	Low	41.38%	34.48%	24.14%
	Middle	44.55%	22.77%	32.67%
	High	62.79%	11.63%	25.58%
TV	Low	46.75%	19.48%	33.77%
	Middle	51.61%	25.81%	22.58%
	High	47.06%	20.59%	32.35%
Total		48.55%	21.97%	29.48%

TABLE 7. Data of real production sample.

Line area	n	t _{min}	t _{max}	sum(t _i)	ct	Station number
Frontend assembly	70	3	45	813	80	12

TABLE 8. Optimization results of Frontend assembly.

Algorithm	Station number	Nodes	Iterations	CPU
Classic B&P	11	3	82	37.421
3-stage B&P	11	2	58	12.884

integer programming, which is the 3rd stage in the algorithm, solved 29.48% of the instances sets optimally.

Additionally, the properties of the precedence graph of the instance have an impact on the operation process. The data sets are classified into three categories, the Number of Tasks (NT), the Order Strength (OS) and the Time Variability ratio (TV), depending on different characteristics to study their influence: (i): Low-NT ($0 \leq NT < 30$), Middle-NT($30 \leq NT < 70$), High-NT($70 \leq NT < 300$). (ii): Low-OS ($0 \leq OS < 30$), Middle-OS($30 \leq OS < 70$), High-OS($70 \leq OS < 100$). (iii): Low-TV($5 \leq TV < 15$), Middle-TV($15 \leq TV < 70$), High-TV($70 \leq TV < 600$). In sets of Low-NT, most (73.81%) instances could get the optimal solution at the 1st stage. And that percentage decreases in Middle-NT (34.69%) and High-NT (43.90%) sets. The percentage value of optimal solutions obtained at 2nd stage for Middle-NT (28.57%) and High-NT (23.17%) sets are more than that for Low-NT (11.90%) set. Also this trend is accordant with the percentage values of 3rd stage. The effectiveness of 3rd stage is higher than the 2nd stage in all three different NT sets. As the order strength increases, the optimal solution obtained at the 1st stage becomes more. However, the percentage of optimal result getting from the 2nd stage decreases when the order strength

TABLE 9. Experiments results of data set of Talbot.

Name	ct	UB	LB1	Z*	Classic Branch-and-Price				3-Stage Branch-and-Price			
					Final Best	Nodes	Iterations	CPU	Final Best	Nodes	Iterations	CPU
Mertens	8	6	4	5	5	3	8	0.569	5	1	3	0.145
Jaeschke	10	5	4	4	4	5	45	2.253	4	1	7	0.267
Jackson	7	9	7	8	8	1	5	0.5	8	1	3	0.17
Jackson	10	6	5	5	5	5	43	2.165	5	1	5	0.213
Mitchell	15	9	7	8	8	15	241	48.372	8	1	34	3.058
Heskiaoff	216	6	5	5	5	3	138	30.377	5	2	40	3.182
Heskiaoff	342	4	3	3	3	9	603	122.445	3	8	563	67.003
Sawyer	25	15	13	14	15	7	291	109.427	14	1	55	6.529
Sawyer	27	14	12	13	13	15	585	192.896	13	1	46	4.839
Sawyer	36	11	9	10	10	7	456	159.584	10	1	73	10.723
Sawyer	41	10	8	8	9	11	578	113.959	9	11	495	51.968
Kilbrid	57	11	10	10	11	15	1726	974.083	10	6	400	91.487
Tonge	176	23	20	21	23	35	2884	3144.29	21	15	1964	1356.845
Arcusl	5048	17	15	16	16	5	517	367.896	16	4	204	105.307
Arcusl	6842	13	11	12	12	5	592	825.165	12	2	100	51.425

TABLE 10. Experiments results of data set of Hoffmann.

Name	ct	UB	LB1	Z*	Classic Branch-and-Price				3-Stage Branch-and-Price			
					Final Best	Nodes	Iterations	CPU	Final Best	Nodes	Iterations	CPU
Sawyer	27	14	12	13	13	15	585	192.896	13	1	46	4.839
Sawyer	33	12	10	11	11	11	521	164.186	11	2	95	10.743
Sawyer	36	11	9	10	10	7	456	159.584	10	1	73	10.723
Sawyer	41	10	8	8	9	11	578	113.959	9	11	495	51.968
Tonge	160	25	22	23	25	43	4412	1746.174	23	4	672	441.319
Tonge	168	25	21	22	24	15	1782	711.837	22	1	207	139.696
Tonge	185	21	19	20	21	13	1678	1291.687	20	2	473	333.216
Tonge	195	20	18	19	20	9	1071	846.873	19	1	247	174.678
Tonge	220	18	16	17	18	5	998	1411.125	17	14	2116	1278.591
Tonge	234	17	15	16	16	3	120	46.558	16	2	149	47.324
Arcusl	3786	22	20	21	22	26	4253	4167.398	21	12	2527	2038.536
Arcusl	6883	13	11	12	12	3	101	65.545	12	2	100	51.766
Arcus2	6837	25	22	23	24	5	1250	3078.478	24	3	403	471.613

is high. In both Middle-OS and High-OS sets, more optimal solutions, 32.67% to 22.77% and 25.58% to 11.63%, are obtained at 3rd stage than at 2nd stage. Nearly half of instances obtain their optimal results at the 1st stage in three different TV sets. The percentage of the 2nd stage and the 3rd stage changes little from the Low-TV to High-TV sets. The influence of the time variability ratio did not affect the operation results as much as the number of task and the order strength.

In order to assess the efficiency of the proposed three-stage branch-and-price algorithm, the numerical tests of a classic branch-and-price algorithm are done on the benchmark instance sets. We list the computational results whose upper bound is not equal to the best-known solutions in Table 9, Table 10, and Table 11. Since the solving process did not enter into the column generation procedure if the initial heuristics could find the best value of solution in a short time. Results of the classic branch-and-price algorithm and the three-stage branch-and-price algorithm are both reported.

Table 9 presents the results of 15 instances of the data set of Talbot. Table 10 shows the detailed results for 13 instances of the Hoffmann set. Results of 51 instances of the Scholl set are listed in Table 11.

For standard branch-and-price algorithm, there are 38 instances which did not achieve the optimal solution. Using three-stage branch-and-price algorithm, only 4 instances got the final solution with a gap of one station to the known best value. There is one instance of Gunther data set (ct = 49) that obtained the optimal solution which is superior to the known best value. The three-stage branch-and-price algorithm is more effective than the standard branch-and-price algorithm, especially for the instances which belong to the High-NT set.

The three-stage branch-and-price algorithm can prune the branching nodes effectively exploring fewer nodes than the classic branch-and-price algorithm. The number of column generation iteration also decreases in the three-stage branch-and-price algorithm. These result in a shorter computing time. There are only 8 of the total instances whose CPU time consumed in the three-stage branch-and-price algorithm is longer than that of the classic branch-and-price algorithm.

B. REAL PRODUCTION CASE

An industry case study for the frontend part assembly of vehicle on the assembly line in BMW Brilliance Automotive Ltd. is conducted to validate our algorithm. The frontend

TABLE 11. Experiments results of data set of Scholl.

Name	<i>ct</i>	UB	LB1	Z*	Classic Branch-and-Price				3-Stage Branch-and-Price			
					Final Best	Nodes	Iterations	CPU	Final Best	Nodes	Iterations	CPU
Buxey	27	14	12	13	14	13	415	155.059	13	1	46	7.595
Buxey	30	13	11	12	12	13	505	185.763	12	1	59	9.907
Buxey	33	12	10	11	11	5	253	94.309	11	1	62	10.311
Buxey	36	11	9	10	10	9	617	159.31	10	1	60	10.188
Lutz1	1414	12	10	11	11	3	100	15.102	11	1	24	3.134
Lutz1	1572	11	9	10	11	7	249	86.833	10	2	99	20.136
Gunther	41	15	12	14	14	5	184	70.042	14	4	121	21.043
Gunther	44	13	11	12	12	7	346	133.763	13	11	205	31.954
Gunther	49	12	10	11	11	17	710	305.24	10	1	83	19.895
Gunther	61	10	8	9	9	5	353	155.371	9	2	204	52.86
Gunther	69	9	7	8	8	3	273	128.93	8	2	196	52.734
Warnecke	54	36	29	31	35	25	1259	1121.97	31	1	67	30.225
Warnecke	56	32	28	29	30	7	302	85.121	29	1	90	41.672
Warnecke	58	32	27	29	29	27	1022	833.718	29	2	119	49.298
Warnecke	60	30	26	27	30	17	995	971.398	27	1	84	24.776
Warnecke	62	30	25	27	30	21	1265	1230.578	27	1	78	35.53
Warnecke	65	29	24	25	29	27	1503	1410.5	25	1	96	46.359
Warnecke	68	27	23	24	24	31	1604	459.486	24	1	121	63.877
Warnecke	71	26	22	23	23	35	1478	402.94	23	1	80	35.667
Warnecke	74	25	21	22	22	29	2024	1835.062	22	1	105	52.084
Warnecke	78	23	20	21	21	29	1705	473.673	21	1	150	90.354
Warnecke	82	22	19	20	22	31	1010	248.116	20	1	129	69.739
Warnecke	86	21	18	19	21	47	2271	2312.996	19	1	140	78.268
Warnecke	97	18	16	17	17	51	2190	1330.847	17	2	320	197.574
Wee-mag	28	64	54	63	63	3	13	5.782	63	1	9	3.203
Wee-mag	29	64	52	63	63	1	3	1.944	63	1	6	2.17
Wee-mag	30	64	50	62	62	1	10	4.682	62	1	11	3.858
Wee-mag	31	63	49	62	62	1	4	2.251	62	1	10	3.45
Wee-mag	32	62	47	61	61	1	5	2.628	61	1	6	2.098
Wee-mag	33	62	46	61	61	1	5	2.584	61	1	6	2.1
Wee-mag	35	61	43	60	60	3	16	6.708	60	2	13	4.522
Wee-mag	42	56	36	55	55	1	6	2.775	55	1	16	6.89
Wee-mag	43	51	35	50	50	1	11	4.189	50	1	12	4.693
Wee-mag	46	39	33	34	35	9	516	235.471	34	8	537	294.774
Wee-mag	47	36	32	33	35	25	942	368.761	33	1	123	73.727
Wee-mag	49	33	31	32	33	13	488	172.748	32	6	376	183.033
Wee-mag	50	33	30	32	33	35	1453	557.667	32	20	971	431.613
Wee-mag	52	33	29	31	32	21	755	271.156	31	36	1442	611.123
Lutz2	11	52	45	49	51	13	244	273.084	49	1	103	123.235
Lutz2	12	49	41	44	44	7	336	302.536	44	1	112	132.331
Lutz2	13	44	38	40	41	19	799	689.439	40	1	137	161.122
Lutz2	14	41	35	37	40	39	1841	1578.235	37	2	260	298.951
Lutz2	15	37	33	34	36	31	1469	1198.571	34	4	352	349.756
Lutz3	75	25	22	23	25	7	1203	1936.181	23	3	576	841.972
Lutz3	79	23	21	22	23	31	1526	2455.307	22	3	609	930.247
Lutz3	83	23	20	21	23	26	2154	3470.094	21	3	614	920.81
Lutz3	87	22	19	20	22	15	2940	4729.448	20	11	1733	2267.173
Lutz3	97	19	17	18	19	12	1266	1565.075	18	5	344	288.953
Mukherje	222	21	19	20	21	7	1100	2087.254	20	4	800	820.156
Mukherje	234	20	18	19	20	17	2747	5103.27	19	24	3905	3888.872
Mukherje	248	19	17	18	19	17	2789	4905.645	18	1	200	219.01

pre-assembly line consists of 12 work stations as shown in Fig. 6. As illustrated in Section 3, some parts are assembled on the pre-assembly areas in the assembly shop. After the completion of total assembly of the frontend, this part is delivered to the main assembly line for its installation on the vehicle as indicated in Fig. 4. The production pattern, for which there is one operator working on each station, is same as the model of SALBP we study. Twelve workers are assigned to this section. Apart from movement time (15 seconds) of the conveyor transiting from one station to the next

station, the reference cycle time is set as 80 seconds for the operator. The data of the test instance is shown in Table 7.

Current line balancing solution needs 12 workers to finish the pre-assembly work of frontend parts. Results after optimization show that the station number can decrease to 11 with the cycle time of 80s. Finally, the percentage of the total idle time of test area dropped to 7.61% from 15.31%. We also list the results of our method and that of classic branch-and-price in Table 8. Compared to the standard branch-and-price algorithm, the number of searching nodes is lesser

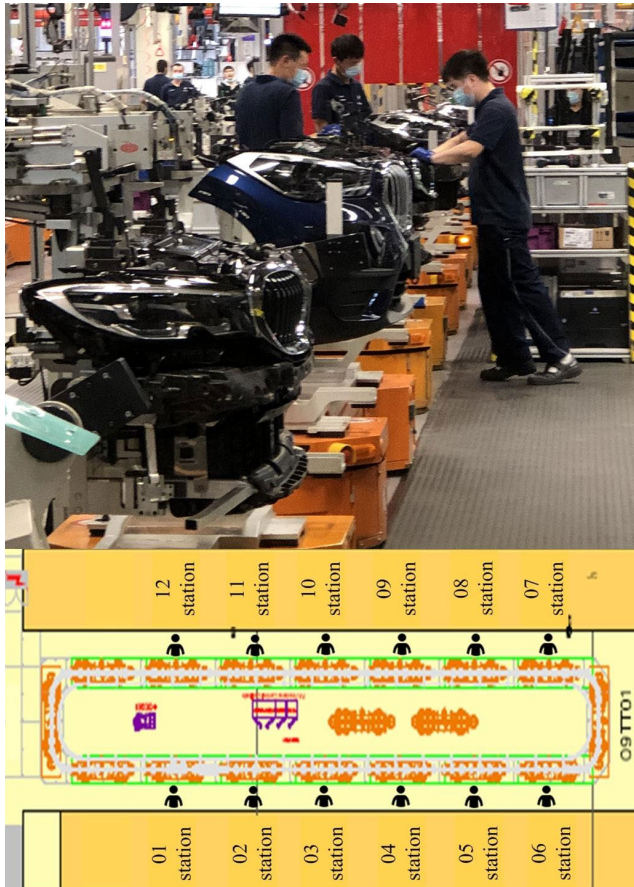


FIGURE 6. Pre-assembly line of the vehicle's frontend part.

in the three-stage branch-and-price algorithm (2 to 3), and its total iteration time is also lesser (58 to 82) which result in a faster computing time (12.884s to 37.421s).

VII. CONCLUSION

In this article, we have reformulated the SALBP-I applying the Dantzig-Wolfe decomposition. Sequence constraints are maintained in the model of the master problem. The three-stage branch-and-price algorithm for solving the SALBP-I is presented. Considering the nature of solutions created through the column generation process, a specific branching strategy is introduced to assure the efficiency of the optimal result searching. Details of the construction and solving the pricing problems are described.

Experiments are conducted on the standard benchmark data sets which cover a wide range of the task scale. The proposed three-stage branch-and-price algorithm obtains competitive results compared to the commercial solver. Three-stage branch-and-price algorithm is also superior to the classic branch-and-price algorithm in the terms of the efficiency of solving and the quality of solution. The analysis of the computational results shows that properties of instances have effects on the optimization algorithm. The number of tasks and the order strength has more impact on the

optimization process than value of time variety ratio. A case study on one production line in BMW Brilliance Automotive Ltd. verifies the effectiveness of the proposed algorithm. Following the principle of production ('allowed workplace', Section_3_A2) the super-larger scale of assembly line balancing in the automotive industry can be decomposed into smaller instances of different areas. Most of these instances belong to the range of scale that the proposed algorithm can deal with. Even under complex industrial conditions, the practical application of our proposed algorithm is still feasible to improve the productivity and reduce the production cost. Although the SALBP-I is a basic issue of assembly line balancing work, insights and comprehension of the proposed method will guide further research works on more complicated instances of assembly line balancing problem.

REFERENCES

- [1] I. Baybars, "A survey of exact algorithms for the simple assembly line balancing problem," *Manage. Sci.*, vol. 32, no. 8, pp. 900–932, 1986.
- [2] A. Scholl and C. Becker, "State-of-the-art exact and heuristic solution procedures for simple assembly line balancing," *Eur. J. Oper. Res.*, vol. 168, no. 3, pp. 666–693, Feb. 2006.
- [3] J. M. Charlton and C. C. Death, "A general method for machine scheduling," *Int. J. Prod. Res.*, vol. 7, no. 3, pp. 207–217, Jan. 1968.
- [4] J. R. Jackson, "A computing procedure for a line balancing problem," *Manage. Sci.*, vol. 2, no. 3, pp. 261–271, Apr. 1956.
- [5] M. Held, R. M. Karp, and R. Shreshian, "Assembly-line balancing—Dynamic programming with precedence constraints," *Oper. Res.*, vol. 11, no. 3, pp. 442–459, Jun. 1963.
- [6] L. Schrage and K. R. Baker, "Dynamic programming solution of sequencing problems with precedence constraints," *Oper. Res.*, vol. 26, no. 3, pp. 444–449, Jun. 1978.
- [7] M. Queyranne, "Bounds for assembly line balancing heuristics," *Oper. Res.*, vol. 33, no. 6, pp. 1353–1359, Dec. 1985.
- [8] A. Bockmayr and N. Pisanuk, "Solving assembly line balancing problems by combining IP and CP," in *Proc. Discrete Math.*, Prague, Czech Republic, 2001. [Online]. Available: <https://arxiv.org/pdf/cs/0106002.pdf>
- [9] A. Pinnoin and W. E. Wilhelm, "A family of hierarchical models for the design of deterministic assembly systems," *Int. J. Prod. Res.*, vol. 35, no. 1, pp. 253–280, Jan. 1997.
- [10] A. Pinnoin and W. E. Wilhelm, "Assembly system design: A branch and cut approach," *Manage. Sci.*, vol. 44, no. 1, pp. 103–118, Jan. 1998.
- [11] M. Peeters and Z. Degraeve, "An linear programming based lower bound for the simple assembly line balancing problem," *Eur. J. Oper. Res.*, vol. 168, no. 3, pp. 716–731, Feb. 2006.
- [12] R. Pastor and L. Ferrer, "An improved mathematical program to solve the simple assembly line balancing problem," *Int. J. Prod. Res.*, vol. 47, no. 11, pp. 2943–2959, Jun. 2009.
- [13] M. Vilà and J. Pereira, "A branch-and-bound algorithm for assembly line worker assignment and balancing problems," *Comput. Oper. Res.*, vol. 44, pp. 105–114, Apr. 2014.
- [14] J. Rubinovitz and G. Levitin, "Genetic algorithm for assembly line balancing," *Int. J. Prod. Econ.*, vol. 41, no. 1, pp. 343–354, 1995.
- [15] Y. K. Kim, Y. J. Kim, and Y. Kim, "Genetic algorithms for assembly line balancing with various objectives," *Comput. Ind. Eng.*, vol. 30, no. 3, pp. 397–409, Jul. 1996.
- [16] I. Sabuncuoğlu, R. Ereli, and M. Tanyer, "Assembly line balancing using genetic algorithms," *J. Intell. Manuf.*, vol. 11, no. 2, pp. 295–310, 2000.
- [17] S. G. Ponnambalam, P. Aravindan, and G. M. Naidu, "A multi-objective genetic algorithm for solving assembly line balancing problem," *Int. J. Adv. Manuf. Tech.*, vol. 16, no. 5, pp. 341–352, Jul. 2003.
- [18] J. Yu and Y. Yin, "Assembly line balancing based on an adaptive genetic algorithm," *Int. J. Adv. Manuf. Technol.*, vol. 48, nos. 1–4, pp. 347–354, Apr. 2010.
- [19] A. Baykasoglu, "Multi-rule multi-objective simulated annealing algorithm for straight and U type assembly line balancing problems," *J. Intell. Manuf.*, vol. 17, no. 2, pp. 217–232, Apr. 2006.

- [20] S. A. Seyed-Alagheband, S. M. T. F. Ghomi, and M. Zandieh, "A simulated annealing algorithm for balancing the assembly line type II problem with sequence-dependent setup times between tasks," *Int. J. Prod. Res.*, vol. 49, no. 3, pp. 805–825, Feb. 2011.
- [21] S. D. Lapiere, A. Ruiz, and P. Soriano, "Balancing assembly lines with tabu search," *Eur. J. Oper. Res.*, vol. 168, no. 3, pp. 826–837, Feb. 2006.
- [22] S. Annarongsri and S. Limnararat, "A hybrid tabu search method for assembly line balancing," in *Proc. 7th Int. Conf. Simulation Modelling Optim.* Hangzhou, China, 2007, pp. 443–448.
- [23] P. Fattahi, A. Roshani, and A. Roshani, "A mathematical model and ant colony algorithm for multi-manned assembly line balancing problem," *Int. J. Adv. Manuf. Technol.*, vol. 53, nos. 1–4, pp. 363–378, Mar. 2011.
- [24] L. K. C. Lai and J. N. K. Liu, "ALBO: An assembly line balance optimization model using ant colony optimization," in *Proc. 5th Int. Conf. Natural Comput.*, Tianjian, China, 2009, pp. 8–12.
- [25] J. P. Arabeyre, J. Fearnley, F. C. Steiger, and W. Teather, "The airline crew scheduling problem: A survey," *Transp. Sci.*, vol. 3, no. 2, pp. 140–163, May 1969.
- [26] K. L. Hoffman and M. Padberg, "Solving airline crew scheduling problems by branch-and-cut," *Manage. Sci.*, vol. 39, no. 6, pp. 657–682, Jun. 1993.
- [27] P. C. Gilmore and R. E. Gomory, "A linear programming approach to the cutting stock problem," *Oper. Res.*, vol. 9, no. 6, pp. 849–859, 1961.
- [28] P. C. Gilmore and R. E. Gomory, "A linear programming approach to the cutting stock problem—Part II," *Oper. Res.*, vol. 11, no. 6, pp. 863–888, Dec. 1963.
- [29] M. L. Balinski and R. E. Quandt, "On an integer program for a delivery problem," *Oper. Res.*, vol. 12, no. 2, pp. 300–304, Apr. 1964.
- [30] F. Cullen, J. Jarvis, and D. Ratliff, "Set partitioning based heuristics for interactive routing," *Networks*, vol. 11, no. 2, pp. 125–144, 1981.
- [31] W. E. Wilhelm, "A column-generation approach for the assembly system design problem with tool changes," *Int. J. Flex. Manuf. Sys.*, vol. 11, no. 2, pp. 177–205, 1999.
- [32] G. B. Dantzig and P. Wolfe, "Decomposition principle for linear programs," *Oper. Res.*, vol. 8, no. 1, pp. 101–111, Feb. 1960.
- [33] L. H. Appelgren, "A column generation algorithm for a ship scheduling problem," *Transp. Sci.*, vol. 3, no. 1, pp. 53–68, Feb. 1969.
- [34] L. H. Appelgren, "Integer programming methods for a vessel scheduling problem," *Transp. Sci.*, vol. 5, no. 1, pp. 64–78, Feb. 1971.
- [35] P. H. Vance, C. Barnhart, E. L. Johnson, and G. L. Nemhauser, "Solving binary cutting stock problems by column generation and branch-and-bound," *Comput. Optim. Appl.*, vol. 3, no. 2, pp. 111–130, May 1994.
- [36] P. H. Vance, C. Barnhart, E. L. Johnson, and G. L. Nemhauser, "Branch-and-price algorithms for the onedimensional cutting stock problem," *Comput. Optim. Appl.*, vol. 9, no. 3, pp. 211–228, 1998.
- [37] M. Savelsbergh, "A branch-and-price algorithm for the generalized assignment problem," *Oper. Res.*, vol. 45, no. 6, pp. 831–841, Dec. 1997.
- [38] W. E. Wilhelm, "A technical review of column generation in integer programming," *Optim. Eng.*, vol. 2, no. 2, pp. 159–200, 2001.
- [39] F. B. Talbot, J. H. Patterson, and W. V. Gehrlein, "A comparative evaluation of heuristic line balancing techniques," *Manage. Sci.*, vol. 32, no. 4, pp. 430–454, 1986.
- [40] T. R. Hoffmann, "Eureka: A hybrid system for assembly line balancing," *Manage. Sci.*, vol. 38, no. 1, pp. 39–47, Jan. 1992.
- [41] A. Scholl, "Data set of assembly line balancing problems," *Schriften zur Quantitativen Betriebswirtschaftslehre*, vol. 16, no. 93, pp. 1–28, 1993.



QIDONG YIN received the B.Eng. degree from Chongqing University, Chongqing, China, in 2011, and the M.Eng. degree from North China Electric Power University, Beijing, China, in 2015. He is currently pursuing the Ph.D. degree with Northeastern University, Shenyang, China.

He participated in BMW Brilliance Automotive Ltd., Ph.D. Promotion Program, in 2015, responsible for the new energy vehicle production project. He has research experiences in the field

of parameter design of power system of electric vehicles and transmission system design of wind turbine. His major research interests include power system design and optimization of electric vehicles, modeling and simulation of automotive manufacturing process for new energy vehicles, integer programming, and optimization methods for large scale problems.



XIAOCHUAN LUO (Member, IEEE) received the M.Eng. and Ph.D. degrees from the Harbin Institute of Technology, Harbin, China, in 1999 and 2002, respectively.

From 2002 to 2004, he held a postdoctoral position at the University of Technology of Troyes, Troyes, France. He is currently a Professor with Northeastern University, Shenyang, China. His current research interests include modeling and optimization of processing industry manufacturing systems, production planning and scheduling, and optimization methods. He is a peer review expert of the National Natural Science Foundation of China, the IEEE Reviewer, IJPR, EJIE, and other international journal reviewers. He was selected into the 2008 Ministry of Educations New Century Excellent Talents Support Program and the Million Talents Project in Liaoning Province, in 2009.

• • •