Computers & Industrial Engineering 60 (2011) 830–839

Contents lists available at ScienceDirect

Computers & Industrial Engineering

journal homepage: www.elsevier.com/locate/caie

Firing sequences backward algorithm for simple assembly line balancing problem of type 1 ☆

Ozcan Kilincci *

Department of Industrial Engineering, Dokuz Eylul University, 35100 Bornova, Izmir, Turkey

ARTICLE INFO

Article history:
Received 17 July 2008
Received in revised form 28 January 2011
Accepted 2 February 2011
Available online 26 February 2011

Keywords:
Assembly line balancing
Petri net
Heuristics
SALBP-1

ABSTRACT

The objective of simple assembly line balancing problem type-1 (SALBP-1) is to minimize the number of workstations on an assembly line for a given cycle time. Since SALBP-1 is NP-hard, many iterative backtracking heuristics based on branch and bound procedure, tabu search, and genetic algorithms were developed to solve SALBP-1. In this study, a new heuristic algorithm based on Petri net approach is presented to solve the problem. The presented algorithm makes an order of firing sequence of transitions from Petri net model of precedence diagram. Task is assigned to a workstation using this order and backward procedure. The algorithm is coded in MATLAB, and its efficiency is tested on Talbot's and Hoffmann's benchmark datasets according to some performance measures and classifications. Computational study validates its effectiveness on the benchmark problems. Also comparison results show that the algorithm is efficiency to solve SALBP-1.

1. Introduction

One of the concerns of the manufacturing companies to select a production system is to maximize production rate or to minimize production cost. Assembly system is a type of production system, and aims to maximize production rate by minimizing the cycle time. A typical assembly system consists of sequential workstations and continuous material handling system connecting with workstations. Component parts are assembled at the workstations on the line to produce a finished product. Each workstation has a set of work elements which are divided to complete assembly operations of the product. One of the most important problems on the assembly lines is assignment of the work elements (or tasks) to the workstations, which is assembly line balancing problem. The assembly line balancing problem is to arrange the work elements at the workstations so that the total time required at each workstation is approximately the same.

The assembly line balancing problem has various versions according to number of products on the line and task time information. According to number of items produced on the line, the line balancing problems are categorized into two groups: single model and multi/mixed model. Task time information enlarges the classification into four groups, such as single model deterministic (SMD), single model stochastic (SMS), multi/mixed model deterministic (MMD), and multi/mixed model stochastic (MMS). SMD and MMD types consist of tasks with constant times, but task times in SMS and MMS are variable. In MMD and MMS, multi model includes production of two or more similar types of products produced separately in batches, whereas two or more similar products are simultaneously produced in mixed models. A detailed classification of the assembly line balancing problems with their solution procedures are given in Erel and Sarin (1998).

The SMD problem, also called the simple assembly line balancing problem (SALBP), has received considerable attention from researchers (Baybars, 1986a; Erel & Sarin, 1998; Scholl & Becker, 2006). SALBP is to assign the tasks to an ordered sequence of stations such that the precedence relations are satisfied and some measure of performance is optimized given a finite set of tasks, each having a fixed performance time, and a set of precedence relations which specify the permissible orderings of the tasks (Baybars, 1986a).

Baybars (1986b) defines SALBP by means of the following assumptions:

- All input parameters are known with certainty.
- A task cannot be split among two or more stations.
- Tasks cannot be processed in arbitrary sequences due to the technological precedence requirements.
- All tasks must be processed.
- All stations under consideration are equipped and manned to process any one of the tasks (i.e., it is assumed, in effect, that the fixed and variable costs associated with all the stations are the same and, therefore, they need not to be considered in the model).

☆ This manuscript was handled by area editor Gursel A. Suer.

* Tel.: +90 232 3881047; fax: +90 232 3887864.
E-mail address: ozcan.kilincci@deu.edu.tr

0360-8352/$ - see front matter © 2011 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cie.2011.02.001

- The task process times are independent of the station at which they are performed and of the preceding or following tasks (i.e., process times are fixed and, furthermore, they are not sequence dependent).
- Any task can be processed at any station (i.e., there are no positional, layout or zoning restrictions).
- The total line is considered to be serial with no feeder or parallel subassembly lines (and, therefore, process times are additive at any station) or any possible interaction of this type is ignored.
- The assembly system is assumed to be designed for a unique model of a single product.

Under these assumptions a typical SALBP is classified according to objectives. Type I of SALBP (SALBP-1) aims to minimize the number of workstations on the line for a given cycle time, whereas the objective of type II of SALBP (SALBP-2) is to minimize the cycle time for a fixed the number of workstations. This study focuses on SALBP-1, which is known as NP-hard (Baybars, 1986a). It was solved by trial and error until the problem was formulated by Salverson (1955). Since then, a variety of schemes have been introduced for solving this problem, such as linear programming, integer programming, dynamic programming, and branch and bound approaches (Baybars, 1986a; Erel & Sarin, 1998).

Since the optimal solution of even a modest size problem is impossible by the exact methods which try to find an optimal solution, a considerable research effort has been spent to develop heuristic approaches. The heuristics are classified into three subclasses (Erel & Sarin, 1998). The first category includes single pass decision rule procedures. These procedures implement a list of processing prioritizing scheme for task assignment based on a single attribute of each task. Maximum Ranked Positional Weight, Maximum Task Time, etc., are the single pass decision rule procedures (Talbot et al., 1986, Erel & Sarin, 1998). Another well-known single pass heuristic is LBHA-1 (Baybars, 1986b). LBHA-1 consists of five phases. In the first four phases, the size of the problem is reduced by utilizing various properties of the problem. The last phase is a single pass heuristic procedure applied on the reduced problem. The second category is heuristic procedures with multiple pass decision rules. These heuristics process multiple single pass solutions and select the best solution. COMSOAL (Computer Method of Sequencing Operations for Assembly Lines) is one of heuristics with multiple pass decision rules (Talbot et al., 1986, Erel & Sarin, 1998). The last category consists of heuristics which attempt to improve a solution or a workstation assignment by some iterative backtracking methods. Branch and Bound procedure is very popular approach to develop the heuristic in this category. FABLE by Johnson (1988), OptPack by Nourie and Venta (1991), EUREKA by Hoffmann (1992), SALOME by Scholl and Klein (1997), AGSA by Sprecher (1999), an enumerative heuristic based on Hoffmann's heuristic by Fleszar and Hindi (2003), and a linear programming based lower bound by Peeters and Degrave (2006) are the iterative backtracking procedures based on branch and bound procedures. Tabu search is another popular method to develop heuristics to solve SALBP-1. Scholl and Voß (1996), Chiang (1998) and Lapierre, Ruiz, and Soriano (2006) proposed the tabu search-based heuristics. Also Genetic Algorithm provides to improve an initial solution. Rubinovitz and Levitin (1995), Kim, Kim, and Kim (1996), Bautista, Suarez, Mateo, and Companys (2000) and Sabuncuoglu, Erel, and Tanyer (2000) presented the iterative backtracking heuristics based on Genetic Algorithm. The heuristics with iterative backtracking procedures are the most efficient heuristics in the literature to solve SALBP-1.

In the current paper, a new heuristic with the single pass decision rule is presented, firing sequence backward (FSb) algorithm. The heuristic based on Petri nets (PNs), which is a powerful tool to analyze and model any discrete event system. The precedence relations of the problem are modeled by PNs. PNs are new research area to solve SALBP-1. In the first study (Kilincci & Bayhan, 2006), a heuristic was presented using the reachability analysis of PNs. Available task to assign current workstation is determined using reachability analysis of the PN model. The second study (Kilincci & Bayhan, 2008) presented a heuristic based on P-invariants, which is a property of PNs. Available task sequence lists are obtained using P-invariants of the PN model. One or more tasks in the list can be assigned to current workstation during any assignment process. In this study, a task sequence list is formed using sequence of firing transitions in the PN model. The task in the list is assigned to the workstation using backward procedure. FSb is a different heuristic from the heuristics presented in the previous studies (Kilincci & Bayhan, 2006, 2008) in view of determination of available tasks. In the first study (Kilincci & Bayhan, 2006), available task to assign current workstation is determined using the reachability analysis. In the second study (Kilincci & Bayhan, 2008), available task to assign current workstation is determined using the P-invariants of the PN model. In this study, available task to assign current workstation is determined using the list which is formed according to sequence of firing transitions in the PN model.

The rest of the paper is organized as follows: In Section 2, an overview of PNs and PN applications in assembly systems are presented. In Section 3, FSb is described and illustrated. In Section 4, performance of FSb is tested on Talbott's and Hoffmann's benchmark datasets according to some performance measures and classifications. To observe efficiency of FSb, its results are compared with results by some heuristics in the literature in Section 4. Comparison section consists of three subsections. FSb and the exact heuristics based on PNs are compared in the first subsection. FSb is compared with well-known single and multi pass heuristics in second subsection. The last subsection includes a comparison between FSb and some iterative backtracking heuristics. The last section includes conclusion and future research directions.

2. Overview of PNs and PN applications in assembly systems

PNs are mathematical and graphical tools used for modeling, formal analysis, and design of discrete event systems. A PN may be defined as a particular kind of bipartite directed graph populated by three types of objects: places, transitions, and directed arcs (Zurawski & Zhou, 1994). In assembly systems, places represent resources such as machines, AGVs, and parts in a buffer. Transitions represent events, tasks or jobs. A transition firing represents an activity. Places and transitions together represent conditions and precedence relations. A token in a place implies that the condition is true. Product information or AGV movement can also be traced with related token movement in the assembly system. In the study by Zhou and Venkatesh (1999), *concepts in manufacturing* and *corresponding PN modeling* is given in Table 1. We also use this table for *concepts in assembly*, since these concepts are also valid for the assembly system.

Places are classified as input place and output place according to direction of arcs that connect them to transitions. Input and output places can represent pre-conditions and post-conditions of transitions. Tokens, which are pictured small solid dots in places, are very useful in order to trace dynamic behavior of the modeled assembly system. The movement or distribution of tokens in places is called PN marking that defines the current state of the modeled system. The marking of a PN model with m places is delineated by a row matrix $\mathbf{M}$. $\mathbf{M}$ includes nonnegative integer numbers, and $M(p)$ represents the number of tokens in the pth place. $\mathbf{M_0}$ is the initial token condition that points out the minimum number of tokens to begin token movement. $\mathbf{M_k}$ denotes the current state of a PN model after the kth firing.

Table 1
Modeling convention with Petri nets.

Concepts in assembly	Petri net modeling
Moving or production lot size	Weight of directed arcs modeling moving or production Kanban
Number of resources, e.g., AGVs, machines, workstations, and robots	The number of tokens in places modeling its availability
Work-in-process	The number of tokens in places modeling the buffers and operations of all machines
Production volume	The number of tokens in places modeling the counter for the number of firing of transitions modeling end of a product
The time of an operation, e.g., setup, processing, and loading	Time delays associated with place or transition modeling the operation
Conveyance or transportation time	Time delays associated with directed arc, place, or transition modeling the operation
System state	Petri net marking (plus timing information for timed Petri net)
Sequence, concurrency, conflict, resource-sharing, etc.	Available Petri net class such as event graph, state machine, and timed Petri net

Token movement is based on *enabling* and *firing* rules (Zurawski & Zhou, 1994). A transition t is *enabled* if each of its input places contain at least the number of tokens, which is equal to the weight of the arc connecting these places to t. *Firing* of the *enabled* transition t is moved the tokens from input places of t to output places of t. The number of tokens moving is equal to the weight of the arc between the places.

Two primary techniques are used to determine the behavior of PNs: the reachability analysis and matrix-equation method. Reachability analysis shows all the markings that can be reached from the initial marking. The matrix-equation method determines any token condition (or marking) in a PN using the incidence matrix. Incidence matrix shows all the connections between transitions and places with weight of arcs. The matrix-equation can be written as

$$\mathbf{M_k} = \mathbf{M_{k-1}} + \mathbf{A}_i \quad \mathbf{k} = 1, 2, \ldots \tag{1}$$

In Eq. (1), $\mathbf{M_k}$ depicts the kth marking of PN, $\mathbf{M_{k-1}}$ denotes the $(k-1)$th marking, and $\mathbf{A}_i$ is ith row of the incidence matrix $\mathbf{A}$. Index i defines the transition firing number.

PNs have been popular methods to solve the various problems of the manufacturing systems in the literature (for a comprehensive review see Moore & Gupta, 1996). PNs can be applied to any area of manufacturing system that can be described graphically like flowchart and that needs some means of representing parallel or concurrent activities (Murata, 1989). Some advantages of modeling of the assembly systems by PNs are following:

- PNs capture the precedence relations and structural interactions of deterministic, stochastic, concurrent and asynchronous events.
- The properties of the assembly system can be observed with token movement. For example conflicts, deadlock points, etc., can be determined easily and efficiently.
- Also capacity overflow condition can be detected by not making any simulation.
- Analysis of the system can be performed without simulation study. Production rate, cycle time, resource utilization, reliability, and performability can be evaluated.
- Bottleneck machines can be identified.
- PNs can be used to implement real-time control systems for a manufacturing system.

Because of these advantages listed above, assembly systems have been one of the popular application areas for PN researchers. Many PN studies focused on modeling and controlling the assembly systems (Adamou, Zerhoni, & Bourjault, 1998; Hsieh, 2004; Ramaswamy, Valavanis, & Barber, 1997; Yu, Yin, Sheng, & Chen, 2003; Zha, Lim, & Fok, 1998), modeling the robotic assembly systems and the robotic tasks (D'souza & Khator, 1997; Yeung & Moore, 2000; Zha, 2000; Zha, Du, & Lim, 2001; Zimmermann & Hommel, 1999), generating the assembly plans and the sequences (Ben-Arieh, Kumar, & Tiwari, 2004; Frey, 2000; Jeng, Lin, & Huang, 1999; Yee & Ventura, 1999), and planning the assembly/disassembly processes (Moore, Gungor, & Gupta, 1998, 2001; Singh, Tiwari, & Mukhopadhyay, 2003; Tang, Zhou, & Caudill, 2001; Tiwari, Sinha, Kumar, Rai, & Mukhopadhyay, 2002; Zha & Lim, 2000; Zussman & Zhou, 1999). Using PNs to solve an assembly line balancing problem is a new research area (Kilincci & Bayhan, 2006, 2008). In this paper, a new heuristic based on PNs is proposed to solve the SALBP-1 considering the advantages of PNs given above. The proposed heuristic will be explained in the next section in detail.

3. FSb Algorithm

The proposed FSb algorithm is a single pass heuristic and uses a list of firing sequences of transitions in the PN model of the precedence relations between tasks.

Transitions in the PN model symbolize tasks. Each transition has exactly one input place and exactly one output place. Input place and output place of any transition explain pre-condition and post-condition of any task for assignment process. The PN model is Ordinary Petri net model in which all the weights are equal to one. Therefore the weight of arcs between places and transitions is always one due to control of assignment. If there is a token in input place of any transition, this transition is enabled. An enabled transition means that the task represented by this transition is available for assignment to current workstation.

After an enabled transition was fired, in other words after the task represented by enabled transition was assigned to the current workstation, one token moves from the input place to the output place. If there is a token in output place of any transition, it is concluded that the task represented by this transition has been assigned to a workstation. To start assignment procedure, source places which are only input places of any transitions have one token. During the assignments, token is moved from a place to a place. When assignments are completed, sink places which are only output places of any transitions have one token.

The proposed FSb consists of two stages. In the first stage, FSb determines firing sequences of transitions using token movement in the net. In the second stage, it assigns the last task in the sequence list to the last workstation using backward procedure if the task provides precedence constraint and time constraint. Flowchart of the algorithm is given in Fig. 1.

Inputs of the proposed algorithm are precedence relations between tasks, task times, and the cycle time. The algorithm obtains the incidence matrix of the PN model from the precedence relations. Initial and the last token conditions are formed from the incidence matrix. Initial token condition and the incidence matrix are required to determine firing sequence of transition in the PN model. FSb finds the sequence list using the reachability analysis and the matrix-equation.

Assignment procedure starts with the last task in the list to assign to the last workstation. At the end of every assignment,

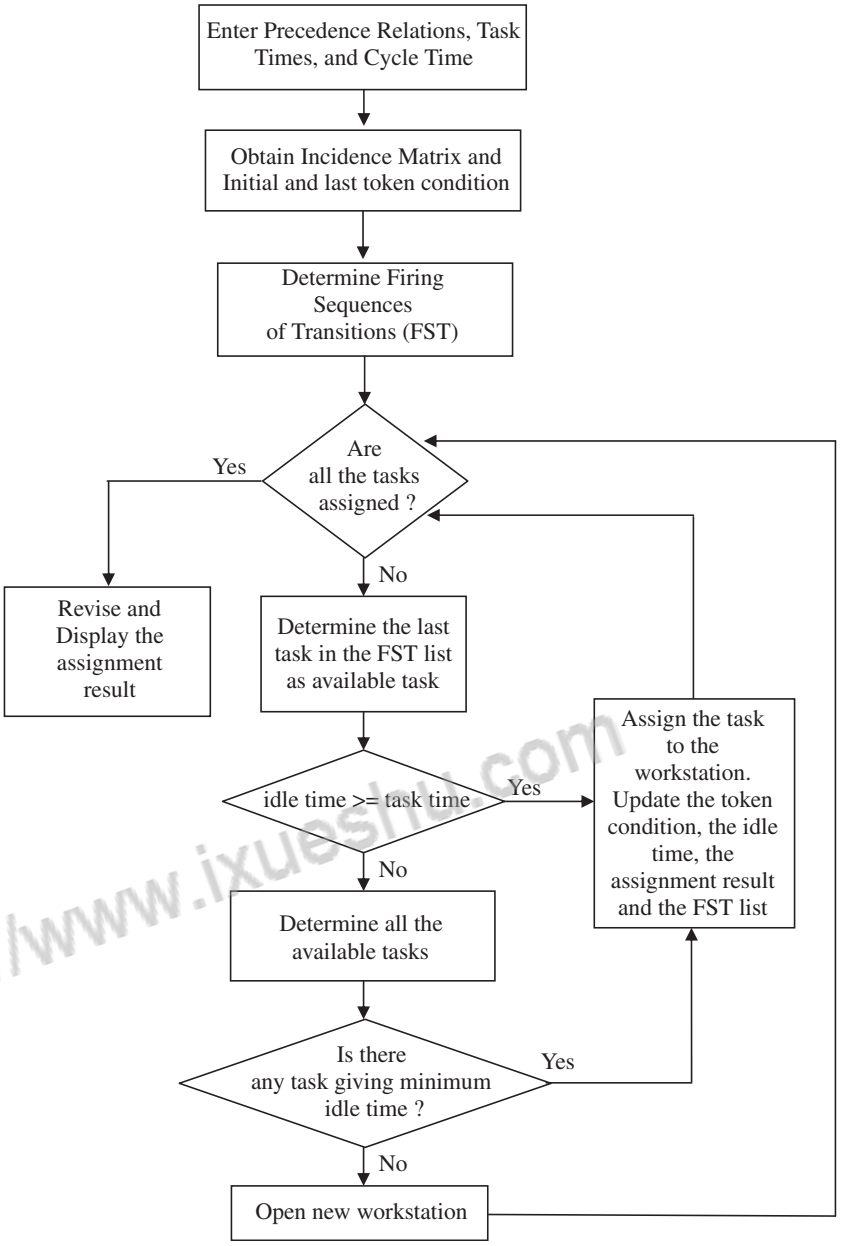

Fig. 1. Flowchart of the FSb algorithm.

token condition is updated using the reachability analysis and the incidence matrix in order to trace assignment procedure. Also assigned task number is deleted from the list. Idle time of the current workstation is recalculated. Procedure is repeated for new last task in the list. If time of the last task in the list is longer than the idle time of the current workstation, next tasks in the list are determined as available task by the heuristic. FSb selects the task minimizing the idle time into available tasks to assign. If no any assignment, new workstation is opened. This procedure is repeated until all the tasks are allocated to the workstations. When the assignments are completed, the assignment result is revised due to the backward procedure. Finally, line efficiency is calculated by using

$$\text{Line efficiency} = \frac{\text{Sum of task times}}{\text{Number of workstations} \times \text{Cycle time}} \quad (2)$$

To further illustrate the implementation of FSb, consider a problem given in Fig. 2. Task numbers are shown in circles, and task times are given next to circles. Desired cycle time is $C = 22$. In the followings, this problem is solved by applying the steps of the algorithm given in Fig. 1.

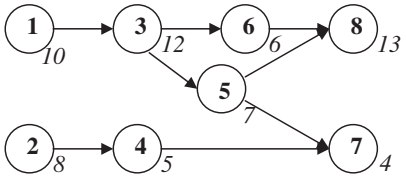

Fig. 2. Precedence relation diagram of the example.

3.1. Inputs

Inputs of the problem are the precedence relations (**PR**), the task times (**t**), and the cycle time (C) given below,

$$\mathbf{PR}=\begin{bmatrix}1&3\\2&4\\3&5\\3&6\\4&7\\5&7\\5&8\\6&8\end{bmatrix},\quad \mathbf{t}=\begin{matrix}t_1&t_2&t_3&t_4&t_5&t_6&t_7&t_8\end{matrix}\\\begin{bmatrix}10&8&12&5&7&6&4&13\end{bmatrix},\quad C=22$$

3.2. Obtaining incidence matrix, initial token condition, last token condition, and firing sequences of transitions

The algorithm converts the precedence relations to the incidence matrix. While the tasks are represented by the transitions, the precedence relations and availability information are symbolized by the places. The incidence matrix (**A**) obtained from the precedence relations is,

$$\mathbf{A}=\begin{matrix}p_1&p_2&p_3&p_4&p_5&p_6&p_7&p_8&p_9&p_{10}&p_{11}&p_{12}\end{matrix}\\\begin{bmatrix}-1&0&1&0&0&0&0&0&0&0&0&0\\0&-1&0&1&0&0&0&0&0&0&0&0\\0&0&-1&0&1&1&0&0&0&0&0&0\\0&0&0&-1&0&0&1&0&0&0&0&0\\0&0&0&0&-1&0&0&1&1&0&0&0\\0&0&0&0&0&-1&0&0&0&1&0&0\\0&0&0&0&0&0&-1&-1&0&0&0&1\\0&0&0&0&0&0&0&0&-1&-1&1&0\end{bmatrix}\begin{matrix}t_1\\t_2\\t_3\\t_4\\t_5\\t_6\\t_7\\t_8\end{matrix}$$

To start the token movement in the PN model, an initial marking ($\mathbf{M_0}$) must be defined. In our example, one token is placed in the source places, p_1 and p_2, in order to begin the token movement. Therefore the first two elements of $\mathbf{M_0}$ are 1.

$$\mathbf{M_0}=\begin{matrix}p_1&p_2&p_3&p_4&p_5&p_6&p_7&p_8&p_9&p_{10}&p_{11}&p_{12}\end{matrix}\\\begin{bmatrix}1&1&0&0&0&0&0&0&0&0&0&0\end{bmatrix}$$

Also the last token condition ($\mathbf{M_L}$) must be determined because it is needed for assignment process with backward procedure. Therefore, the sink places in the net, p_{11} and p_{12}, have one token.

$$\mathbf{M_L}=\begin{matrix}p_1&p_2&p_3&p_4&p_5&p_6&p_7&p_8&p_9&p_{10}&p_{11}&p_{12}\end{matrix}\\\begin{bmatrix}0&0&0&0&0&0&0&0&0&0&1&1\end{bmatrix}$$

Fig. 3 displays the PN model of the precedence relations that we built by using the incidence matrix to trace the token movement easily.

Firing sequences of the transitions (FST) list is determined using the incidence matrix, the task times, and the token condition. Transitions are sequenced according to their ending firing times. The list is,

$$\mathrm{FST}=[2\quad 1\quad 4\quad 3\quad 6\quad 5\quad 7\quad 8]$$

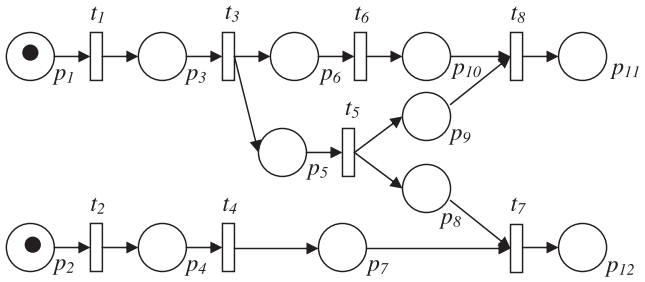

Fig. 3. Petri net model of the precedence relations.

3.3. Assignment procedure

To assign a task to workstation, FSb implements the backward procedure. Therefore the last task in the list is assigned to the last workstation. However, the total number of required workstations in the problem is not known at the beginning. Thus the algorithm starts from workstation 1 to assign the tasks. After all the tasks are assigned, workstation numbers in the result are reorganized.

Task 8 is the first task to assign to workstation 1 in our example because time of task 8 is smaller than the idle time (13 < 22). To determine the token condition after assigning task 8,

$$\mathbf{M_{L-1}}=\mathbf{M_{L-A_8}}=[0\quad 0\quad 0\quad 0\quad 0\quad 0\quad 0\quad 0\quad 0\quad 0\quad 1\quad 1]\\-[0\quad 0\quad 0\quad 0\quad 0\quad 0\quad 0\quad 0\quad -1\quad -1\quad 1\quad 0]$$

$$\mathbf{M_{L-1}}=[0\quad 0\quad 0\quad 0\quad 0\quad 0\quad 0\quad 0\quad 1\quad 1\quad 0\quad 1]$$

New idle time is equal to 9 (idle time − time of task 8 = 22 − 13 = 9).

First assignment result is,

$$[1\quad 8\quad 13\quad 9]$$

where first column defines the workstation number, second column shows the assigned task number, third column explains the assigned task time, and the last column shows the idle time.

Also the assigned task number (8) is deleted from the FST list. Updated FST is

$$\mathrm{FST}=[2\quad 1\quad 4\quad 3\quad 6\quad 5\quad 7]$$

The algorithm continues to assign the tasks to the workstation because all the tasks have not been assigned yet. The second assigned task is task 7 which is the last element in the updated FST list. After assigning task 7, the idle time of workstation 1 is equal to 5. New token condition is,

$$\mathbf{M_{L-2}}=\mathbf{M_{L-1}}-\mathbf{A_7}=[0\quad 0\quad 0\quad 0\quad 0\quad 0\quad 0\quad 0\quad 1\quad 1\quad 0\quad 1]\\-[0\quad 0\quad 0\quad 0\quad 0\quad 0\quad -1\quad -1\quad 0\quad 0\quad 0\quad 1]$$

$$\mathbf{M_{L-2}}=[0\quad 0\quad 0\quad 0\quad 0\quad 0\quad 1\quad 1\quad 1\quad 1\quad 0\quad 0]$$

And the assignment result is updated by adding new assignment to old one.

$$\begin{bmatrix}1&8&13&9\\1&7&4&5\end{bmatrix}$$

FST list is updated by deleting task 7.

$$\mathrm{FST}=[2\quad 1\quad 4\quad 3\quad 6\quad 5]$$

The last task in the FST is task 5 now but time of task 5 is longer than the current idle time (7 > 5). The algorithm searches the available tasks according to $\mathbf{M_{L-2}}$. Tasks 4, 5, and 6 can be made before starting tasks 7 and 8. Task 4 is the best available task in view of minimizing the idle time. If task 4 is assigned to workstation 1, the idle time is equal to zero. After assigning task 4, the algorithm updates the token condition, the idle time, the assignment result, and FST list;

$$\mathbf{M_{L-3}}=\mathbf{M_{L-2}}-\mathbf{A_4}=[0\quad 0\quad 0\quad 0\quad 0\quad 0\quad 1\quad 1\quad 1\quad 1\quad 0\quad 0]\\-[0\quad 0\quad 0\quad -1\quad 0\quad 0\quad 1\quad 0\quad 0\quad 0\quad 0\quad 0]$$

$$\mathbf{M_{L-3}}=[0\quad 0\quad 0\quad 1\quad 0\quad 0\quad 0\quad 1\quad 1\quad 1\quad 0\quad 0]$$

$$\begin{bmatrix}1&8&13&9\\1&7&4&5\\1&4&5&0\end{bmatrix}$$

$$\mathrm{FST}=[2\quad 1\quad 3\quad 6\quad 5]$$

The algorithm opens new workstation after assigning task 4 because the first workstation is full. Task 5, the last task in the list, is assigned to workstation 2. The remaining tasks are assigned to workstations 2 and 3 using similar procedure.

3.4. Revising assignment result

The assignment procedure is completed when all the tasks are assigned to the workstations. The algorithm revised the workstation numbers in the result. Unrevised and revised assignment result is given below,

$$\begin{bmatrix} 1 & 8 & 13 & 9 \\ 1 & 7 & 4 & 5 \\ 1 & 4 & 5 & 0 \\ 2 & 5 & 7 & 15 \\ 2 & 6 & 6 & 9 \\ 2 & 2 & 8 & 1 \\ 3 & 3 & 12 & 10 \\ 3 & 1 & 10 & 0 \end{bmatrix} \quad \begin{bmatrix} 1 & 1 & 10 & 12 \\ 1 & 3 & 12 & 0 \\ 2 & 2 & 8 & 14 \\ 2 & 6 & 6 & 8 \\ 2 & 5 & 7 & 1 \\ 3 & 4 & 5 & 17 \\ 3 & 7 & 4 & 13 \\ 3 & 8 & 13 & 0 \end{bmatrix}$$

Example problem is solved by the proposed algorithm; the eight tasks have been assigned to three workstations (WS1 $= \{1, 3\}$, WS2 $= \{2, 6, 5\}$, WS3 $= \{4, 7, 8\}$), and the line efficiency is 98.48%.

4. Computational experiments

Computational experiments are organized in two subsections. In the first subsection, the proposed algorithm is tested on benchmarking problems in the literature according to some performance measures and classifications. FSb and the exact heuristics in the literature are compared in the second subsection. The algorithm was coded in MATLAB 6.1 for computational experiments.

4.1. Computational results for benchmarking datasets

To test the performance of FSb, Talbot's and Hoffmann's benchmarking problem sets in Scholl and Klein (1999) were used. Information about these benchmarking problems is organized in Table 2 by summarizing from Scholl and Klein (1997). The first nine columns give characteristics of the problem. The number of different cycle times for each set is given in the fifth column. In this column, *T* defines a problem in Talbot's set, and *H* defines a problem in Hoffmann's set. The number in the parenthesis gives the number of the instances for each problem. Each problem is formed for various cycle times. For example, Tonge's problem is formed for five cycle times in Talbot's dataset, whereas twelve cycle times in Hoffmann's set. Therefore there are five instances of Tonge's problem in Talbot's and twelve instances are in Hoffmann's. Ninth column shows the order strengths of the problems in percent. Order strength is a measure of the volume of distinct orderings that are permitted by the specified precedence relations (Erel & Sarin, 1998). The last two columns in the Table 2 display the number of transitions and places in the formed PN model from the precedence diagram of the problem, respectively.

All the problems in the benchmarking datasets are solved by the proposed algorithm, and the results are summarized in Table 3 according to some performance measures. The performance measures examined are the number of optimal solutions, the number of optimal solutions in%, average deviation, average deviation in%, maximum deviation, and maximum deviation in%. The number of optimal solutions is the number of the instances for which an optimal solution is found. Average deviation means average of difference between results by FSb and optimum result, and maximum deviation means maximum difference result by FSb from optimality. Average deviation in% means average relative deviation from the optimality, and maximum deviation in% means maximum relative deviation from the optimality.

In Table 3, the results are classified into three groups such as Talbot's set, Hoffmann's set, and the combination of two sets.

Table 2
Characteristics of the benchmarking problems.

Author	Number of tasks	Min cycle time	Max cycle time	Set	Min task time	Max task time	Sum of task times	Order strength (%)	Number of transitions	Number of places
Mertens	7	6	18	T(6)	1	6	29	52.4	7	10
Bowman	8	20	20	T(1)	3	17	75	75.0	8	11
Jaeschke	9	6	18	T(5)	1	6	37	83.3	9	13
Jackson	11	7	21	T(6)	1	7	46	58.2	11	15
Mansor	11	48	94	T(3)	2	45	185	60.0	11	15
Mitchell	21	14	39	T(6)	1	13	105	71.0	21	31
Heskiaof	28	138	342	T(6)	1	108	1024	22.5	28	42
Sawyer	30	25	75	H(7), T(7)	1	25	324	44.8	30	40
Kilbridge & Wester	45	56	184	T(6), H(6)	3	55	552	44.6	45	70
Tonge	70	160	527	T(5), H(12)	1	156	3510	59.4	70	100
Arcus1	83	3786	10,816	T(7), H(11)	233	3691	75,707	59.1	83	115
Arcus2	111	5755	17,067	T(6), H(14)	10	5689	150,399	40.4	111	178

Table 3
Performance results according to classifications.

Performance measure	Data sets			Problem size			Order	Str.
	Talbot (64 inst.)	Hoffmann (50 inst.)	Combined (101 inst.)	Small (21 inst.)	Medium (31 inst.)	Large (49 inst.)	⩽50% (42 inst.)	>50% (59 inst.)
# of optimal solutions	50	30	72	17	19	36	27	45
# of optimal solutions (%)	78.125	60	71.28713	80.9524	61.2903	73.4694	64.2857	76.2712
Average deviation	0.2188	0.4	0.2871	0.1905	0.3871	0.2653	0.3571	0.2373
Average deviation (%)	3.6081	3.3477	3.3901	4.0873	5.9522	1.4704	4.0661	2.9089
Maximum deviation	1	1	1	1	1	1	1	1
Maximum deviation (%)	33.333	16.667	33.333	33.333	33.333	9.0909	25	33.333

Talbot's set has 64 instances, and Hoffmann's set has 50 instances. Combination of two sets has 101 instances because 13 instances are the same in both sets. In the literature, size of the problem (i.e. the number of tasks) is considered in classifications of the test problems. Therefore, in this table, the problems are also classified into three groups according to their size. The number of the tasks in the small sized problems is up to 21. In the medium sized problems, it changes between 21 and 45. Finally in the large sized problems, the number of tasks is more than 45. The number of the instances is 21 for the small sized group, 31 for the medium sized group, and 49 for the large sized group. In the literature, order strengths of the problems are also considered in classifications of the test problems. In this study, the order strengths of test problems are classified as $\leqslant$50% and >50%. 42 of 101 instances fall into the first class, and 59 instances fall into the second class.

According to datasets the results show that performance of FSb is better in Talbot's dataset. In problem size, FSb produces good solutions in small and large sizes. In order strength classification, results by FSb are better in the problem with high order strength than in other class.

4.2. Comparing FSb to exact heuristics

This subsection is organized to compare the performance of the proposed algorithm based on single pass decision rule with the exact heuristics in the literature. Comparison results are summarized in four subsections. In the first, FSb is compared with the exact heuristics based on PNs. Second study includes a comparison between FSb and well-known single pass and multiple pass heuristics. The last comparison is made between FSb and some of the iterative backtracking procedures.

4.2.1. Comparison between FSb, PNA, and P-in

All the heuristics are tested on Talbot's and Hoffmann's datasets according to some performance measures. Comparison results are given in Table 4. PNA is the heuristic based on reachability analysis (Kilincci & Bayhan, 2006), and *P*-in is the heuristic used P-invariants of the PN model (Kilincci & Bayhan, 2008). FSb is the best heuristic in four of six performance measures, which are number of optimal solutions, number of optimal solution in%, average deviation, and average deviation in%. FSb and PNA have the same performance in other two performance measures. It is say that FSb is the best heuristic based on PNs in the literature to solve SALBP-1.

Table 4
Performance results of the heuristics based on PNs.

Performance measure	PNA	*P*-in	FSb
# of optimal solutions	70	66	72
# of optimal solutions (%)	69.31	65.35	71.29
Average deviation	0.31	0.38	0.29
Average deviation (%)	3.97	4.68	3.39
Maximum deviation	1	2	1
Maximum deviation (%)	33.33	50	33.33

4.2.2. Comparing FSb with single pass and multiple single pass heuristics

FSb is a single pass heuristic developed to solve SALPB-1. To observe the performance of the FSb, well-known single pass and multiple single pass heuristics are tested on benchmark datasets.

WinQSB is a software package developed by Chang (1998). It includes 19 modules to solve various industrial engineering problems in several areas like linear programming, dynamic programming, queuing theory, projects scheduling, forecasting, scheduling, inventory control, facility layout, etc. One of the modules is Facility Location and Layout (FLL). Its "Line Balancing" solver includes of 12 heuristics to solve SALBP-1, up to 1000 tasks. They are Fewest Followers (FF), Fewest Immediate Followers (FIF), First to Become Available (FBA), Last to Become Available (LBA), Longest Process Time (LPT), Most Followers (MF), Most Immediate Followers (MIF), Random (R), Ranked Positional Weight Method (RPWM), and Shortest Process Time (SPT). Also this module consists of COMSOAL, which is a multiple pass heuristic. All eleven heuristics are tested on Talbot and Hoffmann datasets, and the results are given in Table 5.

FSb shows the best performance in the single pass heuristics in all the performance measures. FSb solves more instances to optimality than the other heuristics. Also FSb produces smaller value of average deviation and average deviation (%) than others. In maximum deviation and maximum deviation (%), LPT, MF, and Fsb have the same results. In comparison with FSb and COMSOAL, COMSOAL is better than FSb but the difference between the results by both heuristics are very small.

Another comparison is made between LBHA-1 and FSb. Baybars (1986b) tested LBHA-1 on some problems of the benchmark datasets. Results by LBHA-1 and FSb are given in Table 6. LBHA-1 is the best in three of the six performance measures, and FSb is the best in others. It is observed that LBHA-1 and FSb produce very near results in the first four performance measures. In other two performance measures in which FSb is the best, the difference between the results is bigger.

As a result, FSb shows the best performance in comparison with eleven single pass heuristics. LBHA-1 and FSb have the similar results in the performance measures. Although COMSOAL is a multiple pass heuristic, both of COMSOAL and FSb produce the similar results. The results show that FSb is an efficient heuristic in the single pass and multiple pass heuristics.

Table 6
Comparison with LBHA-1.

Performance measures	LBHA-1	FSb
# of optimum solutions	40	38
# of optimum solutions (%)	81.63	77.55
Average deviation	0.2	0.22
Average deviation (%)	3.85	3.72
Maximum deviation	2	1
Maximum deviation (%)	66.67	33.33

Table 5
Comparison with the heuristics in WinQS.

Performance measures	FF	FIF	FBA	LBA	LPT	MF	MIF	Rand.	RPWM	SPT	COM.	FSb
# of optimum solutions	34	34	40	39	68	64	49	35	70	25	76	72
# of optimum solutions (%)	33.7	33.66	39.6	38.61	67.33	63.4	48.51	34.65	69.307	24.752	75.248	71.29
Average deviation	1.16	0.95	0.772	0.871	0.327	0.37	0.673	0.851	0.3069	1.6337	0.2475	0.29
Average deviation (%)	10.8	9.552	8.536	7.825	4.366	4.83	6.503	8.748	4.7909	14.094	1.9730	3.39
Maximum deviation	4	4	3	3	1	1	3	3	1	8	1	1
Maximum deviation (%)	50	50	50	33.33	33.33	33.3	50	33.33	50	50	20	33.33

Table 7
Comparison with the heuristic based on Genetic algorithm.

Cycle time	Moodie & Young	Tonge MIF	Tonge RC	Tonge BPC	Nevins BBS	Baybars LBHA-1	GA	FSb
83	48	50	50	49	47	47	48	47
86	47	47	48	47	46	46	46	45
89	44	45	46	44	43	43	43	43
92	43	43	44	43	42	42	42	42
95	42	43	43	41	40	40	40	42
170	24	24	24	23	23	23	23	22
173	24	24	24	23	22	23	23	22
176	22	24	23	22	22	23	22	22
179	22	23	23	21	21	22	22	22
182	22	23	22	21	21	22	22	21
346	11	11	12	11	11	11	11	11
349	11	11	11	11	11	11	11	11
364	11	11	11	11	11	11	11	10

Table 8
Comparison with tabu search-based heuristic by Chiang (1998).

Performance measures	BI with TA	BI without TA	FI with TA	FI without TA	FSb
# of optimum solutions	47	58	47	58	48
# of optimum solutions (%)	77.05	95.08	77.05	95.08	78.69
Average deviation	0.26	0.05	0.26	0.05	0.21
Average deviation (%)	2.80	0.36	2.72	0.462	3.24
Maximum deviation	2	1	2	1	1
Maximum deviation (%)	25	10	25	11.11	33.33

4.2.3. Comparing FSb with iterative backtracking heuristics

The iterative backtracking heuristics improve an initial solution to achieve optimal solution. Therefore the iterative backtracking heuristics have better performance in solution quality than the single pass and the multiple pass heuristics, theoretically. For example, the heuristics based on Branch and Bound procedures, i.e. FABLE by Johnson (1988), OptPack by Nourie and Venta (1991), EUREKA by Hoffmann (1992), SALOME by Scholl and Klein (1997), AGSA by Sprecher (1999), etc., are some of the most efficient heuristics to solve SALBP-1. To observe the performance of FSb, it compares with some iterative backtracking heuristics.

The first heuristics is based on Genetic Algorithm by Sabuncuoglu et al. (2000). They compared their heuristic with six heuristics on Tonge's 70 task problem. Tonge's problem is real-life problem from electronics industry and the problem with cycle times $C = 83$, 86, 89, and 95 are known to be difficult to solve (Baybars, 1986b; Sabuncuoglu et al., 2000). Two of the six heuristics, LBHA-1 (Baybars, 1986b) and MIF, are single pass heuristics. Tonge's Random Choice (RC) and Biased Probabilistic Choice (BPC) are multi pass heuristics. Nevins' Best Bud Search (BBS) and Moodie and Young's heuristics are iterative backtracking heuristics (Erel & Sarin, 1998). Results for Tonge's problem for 13 cycle times are given in Table 7.

It is observed that FSb found the best solutions in 85% of cases. It is the highest score in the heuristics (the best value for 11 of 13 instances). Only BBS achieves the best solution for 10 of 13 instances. The best solutions by other heuristics are less than seven. Although BBS gave better result than FSb for cycle times of 95 and 179, FSb has better performance than BBS for cycle times of 86, 170, and 364.

Another comparison is made between FSb and Chang's algorithms which are based on tabu search (Chiang, 1998). They are best improvement with task aggregation (BI with TA), best improvement without task aggregation (BI without TA), first improvement with task aggregation (FI with TA), and first improvement without task aggregation (FI without TA). Comparison results are given in Table 8. It is observed that FSb and the algorithms with task aggregation have approximately the same results. However, the algorithms with task aggregation have better performance than FSb.

Another tabu search-based heuristic (LRS) to solve SALBP-1 is developed by Lapierre et al. (2006). They compared their solutions with Chiang's BI without TA algorithm (Chiang, 1998) using Arcus 1 and 2 problem sets. Comparison results are given in Table 9. FSb produce approximately the same solutions with LRS and BI without TA. FSb is worse than LRS in only Arcus 2 problem for cycle time of 5755. Other solutions by FSB and LSR are the same. In comparison with BI without TA, FSb is better than BI without TA in two cycle times ($C = 5048$ and $C = 8048$) although BI without TA is better than FSb in one cycle time ($C = 5755$). They produce the same solutions for other cycle times.

Also Lapierre et al. (2006) tested their algorithm on Scholl's problem with 297 tasks and 422 precedence constraints, which is known the largest problem in the literature. Comparison results are shown in Table 10. Their algorithm improves an initial solution. The initial solutions for the instances are given in Table 10. It is observed that there is no any difference between solutions by FSb and LSR. However, it must be noted that the results by FSB are better

Table 9
Comparison with heuristic based on tabu search by Lapierre et al. (2006).

Cycle time	Optimum solution	Chiangs'	LRS's	FSb
Arcus 1 problem				
5048	16	17	16	16
5853	14	14	14	14
6842	12	12	12	12
7571	11	11	11	11
8412	10	10	10	10
8898	9	9	9	9
10,816	8	8	8	8
Arcus 2 problem				
5755	27	27	27	28
8847	18	19	18	18
10,027	16	16	16	16
10,743	15	15	15	15
11,378	14	14	14	14
17,067	9	9	9	9

Table 10
Comparison with heuristic by Lapierre et al. (2006) on Schooll's problem.

Cycle time	Initial solution	LRS's	FSb
1394	53	51	51
1422	51	50	50
1452	50	49	49
1483	49	48	48
1515	48	47	47
1548	47	46	46
1584	47	45	45
1620	46	44	44
1659	45	43	43
1699	43	42	42
1742	42	41	41
1787	41	40	40
1834	40	39	39
1883	39	38	38
1935	38	37	37
1991	36	36	36
2049	35	35	35
2111	34	34	34
2177	33	33	33
2247	33	32	32
2322	31	31	31
2402	30	30	30
2488	29	29	29
2580	28	28	28
2680	27	27	27
2787	26	26	26

than the initial solutions by LSR in many instances (16 of 26 instances). It is important that FSb found the same results with LRS by not using any initial solution and any improvement procedure.

In this subsection, some iterative backtracking heuristics based on Genetic Algorithm, Tabu Search, and other approach is compared with FSb to observe its performance. Results show that although FSb is the single pass heuristic which doesn't use any improvement procedure, its efficiency to solve SALBP-1 is similar with some iterative backtracking procedures in the literature.

5. Conclusion

In this study, a heuristic algorithm based on PNs was presented to solve SALBP-1 which aims to minimize number of workstations on an assembly line for a given cycle time. The proposed heuristic obtains a PN model from the precedence relations between tasks. FSb gets a list of firing sequences of transitions (FST) in the PN model. FSb implements an assignment procedure by using FST list and backward procedure.

FSb was coded in MATLAB, and tested on Talbot's and Hoffmann's benchmarking problem sets. Results were analyzed into classifications such as datasets, problem size, and order strength. FSb has achieved the best performance at Talbot's set, the small and large sized problems, and the second class of order strength.

Furthermore, FSb was compared with single pass, multiple pass, and iterative backtracking heuristics. It was observed that the computational results by FSb are encouraging. FSb is better than all well-known single pass heuristics. When FSb was compared with multi pass heuristics, either FSb finds better solutions than others or it finds similar results with others. Another comparison is made between some iterative backtracking heuristics and FSb. Comparison results show that FSb is better than genetic algorithm-based heuristic and FSb has the same performance with tabu search-based heuristics except Chiang's algorithms with task aggregation.

Single pass heuristics are not very efficient as multi pass heuristics and iterative backtracking heuristics as in the literature because of limited search space. Although FSb is a single pass heuristic, its performance is good in comparison results. These results support the efficacy of using FSb to solve SALBP-1. Besides, it should be noted that in benchmark datasets, maximum deviation from optimality is observed only one workstation by FSb produced. Therefore FSb should be preferred as an initial solution for an iterative backtracking heuristic improving the initial solution to increase solution quality.

Further research is planned that the efficiency of the proposed heuristic can be increased by combining with a branch and bound procedure or tabu search. Also FSb can be used to solve simple assembly line balancing problem type-II, which aims to minimize cycle time for fixed number of workstations.

References

Adamou, M., Zerhoni, S. N., & Bourjault, A. (1998). Hierarchical modelling and control of flexible assembly systems using object-oriented Petri nets. *International Journal of Computer Integrated Manufacturing, 11*(1), 18–33.

Bautista, J., Suarez, R., Mateo, M., & Companys, R. (2000). Local search heuristics for assembly line balancing problem with incompatibilities between tasks. In *Proceedings of the 2000 IEEE international conference on robotics & automation, San Francisco, CA* (pp. 2404–2409).

Baybars, I. (1986a). A survey of exact algorithms for the simple assembly line balancing problem. *Management Science, 32*(8), 909–932.

Baybars, I. (1986b). An efficient heuristic method for the simple assembly line balancing problem. *International Journal of Production Research, 24*(1), 149–166.

Ben-Arieh, D., Kumar, R. R., & Tiwari, M. K. (2004). Analysis of assembly operations' difficulty using enhanced expert high-level colored fuzzy Petri net model. *Robotics and Computer-Integrated Manufacturing, 20*(5), 385–403.

Chang, Yih.-Long. (1998). *WinQSB: Decision support software for MS/OM*. New York: John Wiley & Sons, Inc.

Chiang, W. C. (1998). The application of a tabu search metaheuristic to the assembly line balancing problem. *Annals of Operations Research, 77*, 209–227.

D'souza, K. A., & Khator, S. K. (1997). System reconfiguration to avoid deadlocks in automated manufacturing systems. *Computers & Industrial Engineering, 32*(2), 455–465.

Erel, E., & Sarin, S. C. (1998). A survey of the assembly line balancing procedures. *Production Planning Control, 9*(5), 414–434.

Fleszar, K., & Hindi, K. S. (2003). An enumerative heuristic and reduction methods for assembly line balancing problem. *European Journal of Operational Research, 145*(3), 606–620.

Frey, G. (2000). Assembly line sequencing based on Petri net *t*-invariants. *Control Engineering Practice, 8*(1), 63–69.

Hoffmann, T. R. (1992). EUREKA: A hybrid system for assembly line balancing. *Management Science, 38*(1), 39–47.

Hsieh, F. S. (2004). Fault-tolerant deadlock avoidance algorithm for assembly process. *IEEE Transactions on Systems, Man and Cybernetics – Part A: Systems and Humans, 34*(1), 65–79.

Jeng, M. D., Lin, C. S., & Huang, Y. S. (1999). Petri net dynamics-based scheduling of flexible manufacturing systems with assembly. *Journal of Intelligent Manufacturing, 10*(6), 541–555.

Johnson, R. V. (1988). Optimally balancing large assembly lines with FABLE. *Management Science, 34*(2), 240–253.

Kilincci, O., & Bayhan, G. M. (2006). A Petri net approach for simple assembly line balancing problems. *International Journal of Advanced Manufacturing Technology, 30*(11–12), 1165–1173.

Kilincci, O., & Bayhan, G. M. (2008). A P-invariant-based algorithm for simple assembly line balancing problem of type-1. *International Journal of Advanced Manufacturing Technology, 37*(3–4), 400–409.

Kim, Y. K., Kim, Y. J., & Kim, Y. (1996). Genetic algorithms for assembly line balancing with various objectives. *Computers & Industrial Engineering, 30*(3), 397–409.

Lapierre, S. D., Ruiz, A., & Soriano, P. (2006). Balancing assembly lines with tabu search. *European Journal of Operational Research, 168*(3), 826–837.

Moore, K. E., Gungor, A., & Gupta, M. S. (1998). A Petri net approach to disassembly process planning. *Computers & Industrial Engineering, 35*(1–2), 165–168.

Moore, K. E., Gungor, A., & Gupta, S. M. (2001). Petri net approach to disassembly process planning for products with complex AND/OR precedence relationships. *European Journal of Operational Research, 135*(2), 428–449.

Moore, K. E., & Gupta, S. M. (1996). Petri net models of flexible and automated manufacturing systems: A survey. *International Journal of Production Research, 34*(11), 3001–3035.

Murata, T. (1989). Petri nets: properties, analysis and applications. *Proceedings of IEEE, 77*(4), 541–580.

Nourie, F. J., & Venta, E. R. (1991). Finding the optimal line balances with OptPack. *Operations Research Letters, 10*, 165–171.

Peeters, M., & Degrave, Z. (2006). An linear programming based lower bound for the simple assembly line balancing problem. *European Journal of Operational Research, 168*(3), 716–731.

Ramaswamy, S., Valavanis, K. P., & Barber, S. (1997). Petri net extensions for the development of MIMO net models of automated manufacturing systems. *Journal of Manufacturing Systems, 16*(3), 175–191.

Rubinovitz, J., & Levitin, G. (1995). Genetic algorithm for assembly line balancing. *International Journal of Production Economics, 41*(1–3), 343–354.
Sabuncuoglu, I., Erel, E., & Tanyer, M. (2000). Assembly line balancing using genetic algorithms. *Journal of Intelligent Manufacturing, 11*(3), 295–310.
Salveson, M. E. (1955). The assembly line balancing problem. *Journal of Industrial Engineering, 6*, 18–25.
Scholl, A., & Becker, C. (2006). State-of-the-art exact and heuristic solution procedures for simple assembly line balancing. *European Journal of Operational Research, 168*(3), 666–693.
Scholl, A., & Klein, R. (1997). SALOME: A bidirectional branch-and-bound procedure for assembly line balancing. *INFORMS Journal on Computing, 9*(4), 319–334.
Scholl, A., & Klein, R. (1999). Balancing assembly lines effectively – A computational comparison. *European Journal of Operational Research, 114*(1), 50–58.
Scholl, A., & Voß, S. (1996). Simple assembly line balancing-heuristic approaches. *Journal of Heuristics, 2*(3), 217–244.
Singh, A. K., Tiwari, M. K., & Mukhopadhyay, S. K. (2003). Modelling and planning of the disassembly processes using an enhanced expert Petri net. *International Journal of Production Research, 41*(16), 3761–3792.
Sprecher, A. (1999). A competitive branch-and-bound algorithm for the simple assembly line balancing problem. *International Journal of Production Research, 37*(8), 1787–1816.
Talbot, F. B., Patterson, J. H., & Gehrlein, W. V. (1986). A comparative evaluation of heuristic line balancing techniques. *Management Science, 32*, 430–454.
Tang, Y., Zhou, M. C., & Caudill, R. J. (2001). An integrated approach to disassembly planning and demanufacturing operation. *IEEE Transactions on Robotics and Automation, 17*(6), 773–783.
Tiwari, M. K., Sinha, N., Kumar, S., Rai, R., & Mukhopadhyay, S. K. (2002). A Petri net based approach to determine the disassembly strategy of a product. *International Journal of Production Research, 40*(5), 1113–1129.
Yee, S. T., & Ventura, J. A. (1999). A Petri net model to determine optimal assembly sequences with assembly operation constraints. *Journal of Manufacturing Systems, 18*(3), 203–213.
Yeung, W. H. R., & Moore, P. R. (2000). Genetic algorithms and flexible process planning in the design of fault tolerant cell control for flexible assembly systems. *International Journal of Computer Integrated Manufacturing, 13*(2), 157–168.
Yu, J., Yin, Y., Sheng, X., & Chen, Z. (2003). Modelling strategies for reconfigurable assembly systems. *Assembly Automation, 23*(3), 266–272.
Zha, X. F. (2000). An object-oriented knowledge based Petri net approach to intelligent integration of design and assembly planning. *Artificial Intelligence in Engineering, 14*(1), 83–112.
Zha, X. F., Du, H., & Lim, Y. E. (2001). Knowledge intensive Petri net framework for concurrent intelligent design of automatic assembly systems. *Robotics and Computer Integrated Manufacturing, 17*(5), 379–398.
Zha, X. F., & Lim, S. Y. E. (2000). Assembly/disassembly task planning and simulation using expert Petri nets. *International Journal of Production Research, 38*(15), 3639–3676.
Zha, X. F., Lim, S. Y. E., & Fok, S. C. (1998). Integrated knowledge-based Petri net intelligent flexible assembly planning. *Journal of Intelligent Manufacturing, 9*(3), 235–250.
Zhou, M. C., & Venkatesh, K. (1999). *Modeling, simulation, and control of flexible manufacturing systems: A Petri net approach*. World Scientific Publishing Co. Pte. Ltd.
Zimmermann, A., & Hommel, G. (1999). Modelling and evaluation of manufacturing systems using dedicated Petri nets. *International Journal of Advanced Manufacturing Technology, 15*(2), 132–138.
Zurawski, R., & Zhou, M. (1994). Petri nets and industrial applications: A Tutorial. *IEEE Transactions on Industrial Electronics, 41*(6), 567–581.
Zussman, E., & Zhou, M. C. (1999). A methodology for modeling and adaptive planning of disassembly processes. *IEEE Transaction on Robotics and Automation, 15*(1), 190–194.