Resource Constrained Assembly Line Balancing Problem
Solved with Ranked Positional Weight Rule

Dr. Hsiu-Hsueh Kao (Correspondence author)
Department of Business Administration, Tamkang University
No.151, Yingzhuan Rd., Danshui, New Taipei City, TAIWAN
Tel: +886-2-2621-5656 E-mail: simonkao@mail.tku.edu.tw

Dr. Din-Horng Yeh
Department of Business Administration, National Chung Cheng University
No.168, University Rd., Min-Hsiung Chia-Yi, TAIWAN
Tel: +886-5-272-0411 E-mail: bmadhy@ccu.edu.tw

Dr. Yi-Hsien Wang
Department of Banking and Finance, Chinese Culture University
No. 55, Hwa-Kang Road, Yang-Ming-Shan, Taipei, TAIWAN
Tel: +886-2-2861-0511 E-mail: holland@mail2000.com.tw

Abstract: We study a resource constrained assembly line balancing problem (RCALBP) presented by Ağpak and Gökçen, who developed a 0-1 integer programming model to find the optimal solution. However, this model is inefficient in solving large-scale problems. In this paper, we propose a simple efficient heuristic that is based on the widely-used ranked positional weight (RPW) rule. The example given by Ağpak and Gökçen is used for illustration, and numerical results of sample problems selected from the literature are given to show the effectiveness of the proposed heuristic.

JEL Classifications: C44, C61, D85, E23
Keywords: Assembly line balancing; Resource-constrained; Heuristic; Ranked positional weight

1. Introduction

The assembly line balancing problem (ALBP) has been studied extensively since the pioneer work of Bryton (1954) and Salveson (1955). Numerous techniques developed for the ALBP have been proven useful in various assembly industries such as manufacturing, automobiles, consumer electronics, etc. In the traditional straight-line ALBP, a finite set of tasks for the assembly line is given including the associated processing times and precedence relationships that define the permissible ordering of tasks; the objective is to assign the set of tasks to successive workstations in order to meet specified production requirement, in which the number of workstations needed is minimized (Ghosh and Gagnon (1989)). During past decades, various new ALBP with different concepts have been studied (such as parallel, two-sided, U-type, mixed-model, flexible assembly line, etc.). Many solution techniques, including optimal or heuristic methods, have been developed (see Baybars (1986); Ghosh and Gagnon (1989); Erel and Sarin (1998); Becker and Scholl (2006) for extensive surveys). Recently, some researchers studied the so-called assembly system design problems (ASDP) in such a way that the objective was to optimize an economic criterion (e.g., total cost) with machine selection (see for example, Nicosia et al. (2002); Yamada
and Matsui (2003)). However, only limited researches discuss the situation of resource constraints that arise often in practice when balancing an assembly line.

In the real-world ALBP, different types of resources (such as machines and workers) are needed to perform the processing requirements of tasks. Recently, Ağpak and Gökçen (2005) pointed out that the issue of line balancing with limited resources has always been a serious problem in industry. In their paper, Ağpak and Gökçen (2005) considered a resource-constrained assembly line balancing problem (RCALBP) in which they considered two types of resource-constrained cases in performing the processing requirements of tasks. Unlike most ALBP that aimed to minimize the number of workstations needed, the objective of the RCALBP was to balance the assembly line with the minimum number of resources needed. They developed a 0-1 integer programming model to find the optimal solution; just like the inherent disadvantage of most exact solution techniques, their model was inefficient in solving large-scale problems.

In this paper we consider the same RCALBP given by Ağpak and Gökçen (2005), and the objective of this paper is to propose an efficient heuristic to solve the RCALBP effectively. The proposed heuristic is based on the widely-used ranked positional weight (RPW) rule developed by Helgeson and Birnie (1961). The reason of using the RPW rule is straightforward because it is reported to be efficient, effective, and easy to implement in practice. We use the example given by Ağpak and Gökçen (2005), along with several sample problems appeared in the literature, to show the effectiveness of our proposed heuristic. The paper is organized as follows: descriptions of the traditional ALBP and the RCALBP are given in section 2; the proposed heuristic is described in section 3, followed in section 4 by the example given by Ağpak and Gökçen (2005) for illustration; section 5 summarizes numerical results of sample problems selected from the literature; conclusion and future research are addressed in the last section.

2. The Assembly Line Balancing Problem (ALBP)

2.1 Traditional ALBP

Conceptually, an assembly line consists of a set of tasks that are assigned to successive workstations. Tasks are given with their processing times and precedence relationship among them. Products to be assembled are processed along the line through each workstation in which resources are used to perform the tasks assigned. The purpose of the traditional ALBP is to resolve the issue of task assignment to successive workstations in order to meet production requirements, without violating the given precedence relationship among tasks. The following definition of the traditional ALBP is modified from Gutjahr and Nemhauser (1964).

Given a finite set $S$ of tasks to be assigned, a positive real valued function $T$ defined on $S$ representing processing times of tasks, and a partial order $P$ defined on $S$ denoting precedence relationship among tasks. Also, let $C$ be the cycle time obtained from production requirements (which is the upper bound of the total processing time of tasks assigned to a workstation), and let $N$ be the number of workstations needed in the assembly line. The objective of the traditional ALBP is to group the set of tasks into subsets $S_i \subseteq S$, $i = 1,...,N$, in order to fulfill the cycle time requirement by minimizing the number of workstations needed. Conventionally, subset $S_i$ is usually called workstation $i$, and task $x$ is said to be assigned to workstation $i$ if $x \in S_i$. Let $T(x)$ be the processing time of task $x$, and $T(S_i)$ be the total processing time of tasks assigned to workstation $i$. The grouping of subsets $S_i$ is feasible if the following conditions are satisfied:

\[(1) \quad \bigcup_{i=1}^{N} S_i = S.\]
(2) \[ S_i \cap S_j = \emptyset, \ i \neq j. \]

(3) \[ T(S_i) = \sum_{x \in S_i} T(x) \leq C, \ i = 1, ..., N, \]

(4) If \( xPy \) (i.e., \( x \) precedes \( y \)) and \( x \in S_i, \ y \in S_j \), then \( i \leq j \).

Conditions (1) and (2) simply say that all the tasks have to be assigned, and each task is assigned to one and only one workstation; condition (3) ensures that, for any workstation, the total processing time of tasks assigned does not exceed the cycle time, while condition (4) maintains precedence relationship not to be violated. As stated above, the objective of the traditional ALBP is to resolve the issue of task assignment with the minimum number of workstations needed.

### 2.2 Resource-constrained ALBP (RCALBP)

The problem studied in this paper differs from the traditional ALBP in that, for any workstation, different types of resources are required in performing the tasks assigned to the workstation. In practice, examples of resources include tools, machines, workers, etc. The definition of the resource-constrained assembly line balancing problem (RCALBP), originally proposed by Ağpak and Gökçen (2005), is described as follows. Consider the above traditional ALBP with a given cycle time and an estimated maximum number of workstations (an obvious upper bound is the number of tasks in the assembly line). Assume that, while assigning tasks to each workstation, different types of resources are also required in performing the tasks assigned. Instead of minimizing the number of workstations needed, the objective of the RCALBP is to resolve the issue of task assignment in order to fulfill the cycle time requirement in such a way that the number of resources needed is minimized as possible. In this paper we consider the case in which each task is performed by one single type of resource (the so-called Type I RCALBP by Ağpak and Gökçen (2005)); for Type II RCALBP (in which each task is allowed to be performed by more than one type of resource), extension of the proposed heuristic is straightforward.

### 3. The Proposed Heuristic

In this section we present the proposed heuristic aiming to overcome computational inefficiency of the 0-1 integer programming model developed by Ağpak and Gökçen (2005). The heuristic proposed is based on the ranked positional weight (RPW) rule developed by Helgeson and Birnie (1961), the procedure starts with the first station (\( k = 1 \)). The following stations are considered successively. In each iteration, a task with highest priority which is assignable to the current station \( k \) is selected and assigned. When no additional task is assignable to station \( k \), it is closed, and the next station (\( k+1 \)) is opened. In which positional weight of a task is defined to be the sum of task times for the task itself and all its following tasks. The motivation for task assignment of the proposed heuristic is described as follows. Since the objective of the RCALBP is to minimize the number of resources needed, it is intuitive that the tasks performed by the same type of resource should be assigned to the same workstation as possible in order to reduce the number of resources needed. Therefore, whenever a new workstation is created for task assignment, we apply the RPW rule to select the first task from a set of eligible tasks (a task is said to be eligible if the assignment of this task does not violate the cycle time requirement and precedence relationship); if there are still eligible tasks left after assigning the first task, we reapply the RPW rule to select a task from the remaining eligible tasks that require the same type of resource as that of the first task. Task assignment proceeds until there is no eligible task remaining to be assigned. The procedure of the proposed heuristic and notations used are summarized in the following, along with the flow chart of the proposed heuristic as shown in Figure 1 on the next page.
Procedure of the proposed heuristic:

Initialization:
Set $S_u = S$; $i = 1$

New workstation $i$:
Set $S_i = \emptyset$

First task assignment:
Apply the RPW rule to select $y$ from the set of eligible tasks $S_u$;
Set $S_i = \{y\} \cup S_i$, $S_u = S_u / \{y\}$

Resource constraint:
Modify $S_u$

$S_u \neq \emptyset$ with some tasks using the same type of resource?

No

All the tasks have been assigned?

No

Yes

Stop

Yes

Apply the RPW rule to select $z$ from $S_u$;
Set $S_i = \{z\} \cup S_i$,
$S_u = S_u / \{z\}$, $S_u = S_u / \{z\}$

Figure 1. Flow chart of the proposed heuristic
Notations:

- \( C \): the given cycle time
- \( N \): an estimated maximum number of workstations
- \( S \): the set of tasks in the assembly to be assigned to workstations
- \( S_u \): the set of tasks unassigned
- \( S_a \): the set of eligible tasks to be assigned
- \( S_i \): the set of tasks assigned to workstation \( i \), \( i = 1, \ldots, N \)
- \( T(x) \): processing time of task \( x \in S \)
- \( T(S_i) \): the total processing time of tasks assigned to workstation \( i \)

Step 0. \((Initialization)\) Let \( S_u \) be the set of remaining unassigned tasks. Initially, since all the tasks in \( S \) are unassigned, we have \( S_u = S \). Let \( i = 1 \).

Step 1. \((New \ \text{workstation})\) Create a new workstation \( i \) for tasks assignment, and let \( S_i = \emptyset \).

Step 2. \((First \ \text{task assignment})\) Let \( S_a \subseteq S_u \) denote the set of eligible tasks that can be assigned to workstation (a task \( x \) is said to be eligible if \( T(x) + T(S_i) \leq C \) and the assignment of \( x \) does not violate precedence relationships). Compute positional weights for the tasks in \( S_a \), and let task \( y \in S_a \) be the one with the maximum positional weight. Break ties using the longest processing time rule; if there is still a tie, then a task is selected arbitrarily. Assign task \( y \) to workstation \( i \) by letting \( S_i = \{ y \} \cup S_i \) and \( S_u = S_u \setminus \{ y \} \).

Step 3. \((Resource \ \text{constraint})\) Modify \( S_a \) as necessarily by adding new possible eligible tasks. Two possible cases may occur, in which a new task is assigned if case 1 is satisfied:

Case 1. \( S_a \neq \emptyset \) and there exist some tasks in \( S_a \) using the same type of resource as that used by the tasks already assigned to workstation. Compute positional weights for these tasks, and let task \( z \) be the one with the maximum positional weight. The same tie-breaking rule in Step 2 is used to break ties. Assign task \( z \) to workstation \( i \) by letting \( S_i = \{ z \} \cup S_i \), \( S_u = S_u \setminus \{ z \} \) and \( S_a = S_a \setminus \{ z \} \). Go to Step 3.

Case 2. \( S_a = \emptyset \) or there exists no task in \( S_a \) using the same type of resource as that used by the tasks already assigned to workstation. In this case no more tasks can be assigned to workstation. Stop the procedure if all the tasks have been assigned; otherwise, let \( i = i + 1 \) and go to Step 1.

4. An Illustrative Example

In this section we use the example given by Ağpak and Gökçen (2005) to illustrate the procedure of the proposed heuristic. Network presentation of the illustrative example is as depicted in Figure 2. And Table 1 on the next page gives processing times and resources needed for each task.
Table 1. Processing times and resources needed for tasks

<table>
<thead>
<tr>
<th>Task</th>
<th>Processing time</th>
<th>Resource type</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>6</td>
<td>A</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>B</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td>A</td>
</tr>
<tr>
<td>4</td>
<td>7</td>
<td>B</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>A</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>B</td>
</tr>
<tr>
<td>7</td>
<td>3</td>
<td>A</td>
</tr>
<tr>
<td>8</td>
<td>6</td>
<td>B</td>
</tr>
<tr>
<td>9</td>
<td>5</td>
<td>A</td>
</tr>
<tr>
<td>10</td>
<td>5</td>
<td>B</td>
</tr>
<tr>
<td>11</td>
<td>4</td>
<td>A</td>
</tr>
</tbody>
</table>

There are totally 11 tasks, and the associated processing times and the types of resources needed are given next to the nodes. For instance, task 3 has processing time of 5 and the resource type needed for task processing is A. The cycle time is assumed to be \( C = 9 \), and the maximum number of workstations is estimated to be \( N = 7 \). The computational results are summarized as follows.

**Initialization**

Step 0. \( S_u = S = \{1,2,3,4,5,6,7,8,9,10,11\} \) and \( i = 1 \).

**Iteration 1**

Figure 2. Illustrative example given by Ağpak and Gökçen [1]

Step 1. Create workstation 1, and let \( S_1 = \emptyset \).

Step 2. We have \( S_a = \{1\} \). Assign task 1 to workstation 1 because it is the only eligible task, and let \( S_1 = \{1\} \) and \( S_u = \{2,3,4,5,6,7,8,9,10,11\} \).

Step 3. The set of eligible tasks is modified as \( S_u = \{2,5\} \). Since only task 5 uses the same resource type A as that of task 1, then task 5 is assigned to workstation 1. Let

\[ S_1 = \{1,5\} \]
Let $\mathcal{S}_t = \{1,5\}$, $\mathcal{S}_u = \{2,3,4,6,7,8,9,10,11\}$ and $\mathcal{S}_a = \{2\}$. Since task 2 uses different type of resource, no more tasks can be assigned to workstation 1. Let $i = 2$ and go to Step 1.

**Iteration 2**

Step 1. Create workstation 2, and $\mathcal{S}_2 = \emptyset$.

Step 2. We have $\mathcal{S}_a = \{2,3,4\}$. Both tasks 2 and 4 have the maximum positional weight, assign task 4 to workstation 2 because it has the longest processing time. Let $\mathcal{S}_2 = \{4\}$ and $\mathcal{S}_u = \{2,3,6,7,8,9,10,11\}$.

Step 3. The set of eligible tasks is modified as $\mathcal{S}_a = \{2\}$. Task 2 is assigned to workstation 2 because it is the only eligible task that uses the same type of resource as that of task 4. Let $\mathcal{S}_2 = \{2,4\}$, $\mathcal{S}_u = \{3,6,7,8,9,10,11\}$ and $\mathcal{S}_a = \emptyset$. Since $\mathcal{S}_a = \emptyset$, no more tasks can be assigned to workstation 2. Let $i = 3$ and go to Step 1.

**Iteration 3**

Step 1. Create workstation 3, and $\mathcal{S}_3 = \emptyset$.

Step 2. We have $\mathcal{S}_a = \{3,6\}$. Both tasks 3 and 6 have the maximum positional weight, assign task 3 to workstation 3 because it has the longest processing time. Let $\mathcal{S}_3 = \{3\}$ and $\mathcal{S}_u = \{6,7,8,9,10,11\}$.

Step 3. The set of eligible tasks is modified as $\mathcal{S}_a = \{6\}$. Since task 7 is the only task that uses the same type of resource as that of task 3, we then assign task 7 to workstation 3. Let $\mathcal{S}_3 = \{3,7\}$, $\mathcal{S}_u = \{6,8,9,10,11\}$ and $\mathcal{S}_a = \emptyset$. Since $\mathcal{S}_a = \emptyset$, let $i = 4$ and go to Step 1.

**Iteration 4**

Step 1. Create workstation 4, and $\mathcal{S}_4 = \emptyset$.

Step 2. We have $\mathcal{S}_a = \{6,9\}$. Task 6 is assigned to workstation 4 because it has the maximum positional weight. Let $\mathcal{S}_4 = \{6\}$ and $\mathcal{S}_u = \{8,9,10,11\}$.

Step 3. The set of eligible tasks is modified as $\mathcal{S}_a = \{8,9\}$. Since task 8 is the only task that uses the same type of resource as that of task 6, it is then assigned to workstation 4. Let $\mathcal{S}_4 = \{6,8\}$, $\mathcal{S}_u = \{9,10,11\}$ and $\mathcal{S}_a = \emptyset$. Since $\mathcal{S}_a = \emptyset$, let $i = 5$ and go to Step 1.

**Iteration 5**

Step 1. Create workstation 5, and $\mathcal{S}_5 = \emptyset$.

Step 2. We have $\mathcal{S}_a = \{9,10\}$. Both tasks 9 and 10 have the maximum positional weights, and both have the same processing times. Ties may be broken arbitrarily; however, since there is only task 11 left and it uses the same type of resource as that of task 9, task 10 is selected to be assigned to workstation 5 (which may result in using less number of resources as possible). Let $\mathcal{S}_5 = \{10\}$ and $\mathcal{S}_u = \{9,11\}$.

Step 3. The set of eligible tasks is modified as $\mathcal{S}_a = \emptyset$. Since $\mathcal{S}_a = \emptyset$, let $i = 6$ and go to Step 1.

**Iteration 6**
Step 1. Create workstation 6, and $S_\phi = \phi$. 

Step 2. We have $S_a = \{9\}$. Assign task 9 to workstation 6 because it is the only eligible task. 
Let $S_\phi = \{9\}$ and $S_a = \{11\}$.

Step 3. The set of eligible tasks is modified as $S_a = \{11\}$. Assign task 11 to workstation 6 because it is the only eligible task and uses the same type of resource as that of task 9. 
Let $S_\phi = \{9, 11\}$, $S_a = \phi$ and $S_a = \phi$. Since $S_\phi = \phi$, the procedure stops because all the tasks have been assigned.

Table 2. Task assignment and resources for Ağpak and Gökçen and the proposed heuristic

<table>
<thead>
<tr>
<th>Workstation</th>
<th>Ağpak and Gökçen</th>
<th>Proposed heuristic</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Tasks</td>
<td>Resources</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>A</td>
</tr>
<tr>
<td>2</td>
<td>2, 4</td>
<td>B</td>
</tr>
<tr>
<td>3</td>
<td>3, 5, 7</td>
<td>A</td>
</tr>
<tr>
<td>4</td>
<td>6, 8</td>
<td>B</td>
</tr>
<tr>
<td>5</td>
<td>10</td>
<td>B</td>
</tr>
<tr>
<td>6</td>
<td>9, 11</td>
<td>A</td>
</tr>
</tbody>
</table>

The resulting task assignment by the proposed heuristic is summarized as shown in Table 2. As we can see in the table, there are 6 workstations in which 3 units of resource type A and 3 units of resource type B are needed (namely, workstations 1, 3, and 6 use the resource type A, and workstations 2, 4, and 5 use the resource type B).

The optimal solution obtained by the 0-1 integer programming model in Ağpak and Gökçen (2005) is also given for comparison. As can be seen in the table, although tasks assignment are different for both methods, the proposed heuristic yields the same number of resource units needed as that of the optimal solution obtained by Ağpak and Gökçen (i.e., 3 units of resource A and 3 units of resource B).

5. Numerical Results of Sample Problems

To illustrate the effectiveness of the proposed heuristic, in this section we apply the heuristic to solve sample problems selected from the literature, and we compare the results of the proposed heuristic with the optimal solutions obtained by using 0-1 integer programming model of Ağpak and Gökçen (2005). Since in the literature there are no similar studies considering both processing time and resource type for each task, we select sample problems from other researches and then modify the problems by randomly giving each task one type of resource required so that they will meet the purpose of this study. Because we also need to obtain the optimal solutions for comparison purpose, we restrict our attention to sample problems of moderate size. The sample problems are selected from Bowman (1960), Buxey (1974), Dar-El et. al. (1979), Gutjahr and Nemhauser (1964), Jackson (1956), Jaeschke (1964), Mertens (1967), Pinto et. al. (1983), Scholl and Klein (1997), and Tonge (1960)).

Numerical results of the proposed heuristic, along with the optimal solutions obtained by the 0-1 integer programming model of Ağpak and Gökçen (2005), are given in Table 3. As we can see in the table that in most sample problems, the proposed heuristic is quite effective because it yields the same number of resource units needed as that of the optimal solutions (except for sample problems from Jackson (1956) and Scholl and Klein (1997), in which one more resource unit is needed for the proposed heuristic).

Table 3. Number of workstations and resources needed for selected sample problems
Two comments are worth noting from the above numerical results of sample problems: Firstly, when assigning tasks to a workstation, the intuition of selecting tasks with the same type of resource seems quite effective for the purpose of reducing the number of resource units needed. As a matter of fact, in this study we have also adapted the same intuition to apply another widely-used rule, the maximum number of followers rule, to solve the selected sample problems. We find from the numerical results that the solutions obtained are the same as that of the proposed heuristic, except that the sequences of task assignment are different for both rules. The reason for such results may arise in the similar concept of task assignment adapted by two rules (i.e., the task chosen is the one with the heaviest workload following it), or may simply because the tuition is indeed effective in solving the RCALBP considered.

Secondly, it can be seen that the results obtained by the proposed heuristic tends to assign those tasks with the same resource type to a workstation. The consequence is that the number of resource units needed is the same as the number of workstations needed in the proposed heuristic. Since the objective of the RCALBP is to minimize the number of resource units needed (not the number of workstations), it appears that the proposed heuristic may obtain the same optimal solution if the number of workstations needed is the same for both methods. The reason for this fact is that if a workstation uses more than one type of resources, without increasing the number of resource units needed, it is possible to partition the workstation into many workstations in which only one type of resource is used in each partitioned workstation.

### 6. Conclusion and Further Research

We consider in this study a resource constrained assembly line balancing problem (RCALBP) proposed recently by Ağpak and Gökçen (2005). As they pointed out that, although resource constrained cases were widely experienced in practice, there has not been sufficient interest in the literature. In their paper, they developed a 0-1 integer programming model to find the optimal solution; however, due to the NP-hard nature of ALBP, their model was inefficient in solving large-scale problems. In this paper we propose an effective heuristic in which we apply the RPW rule, along with an intuitive resource-constrained criterion, to resolve the issue of task assignment when balancing the assembly line. The proposed heuristic is computationally efficient since only calculation of positional weights of tasks is needed. Numerical results of the illustrative example and selected sample problems show that the proposed heuristic is quite effective, compared with the optimal solutions obtained by 0-1 integer programming model of Ağpak and Gökçen (2005).

Although the proposed heuristic may not always give the optimal solution, it efficiently provides an effective task assignment that may serve as a starting balance for further improvements. Further research may include: (i) Investigation of applying other heuristics to
improve the effectiveness; (ii) Extension of RCALBP to consider more practical resource constraints; (iii) Extension of RCALBP to minimize the number of workstations required for a given number of resources; or (iv) Extension of RCALBP to minimize the cycle time for a given number of workstations and resources, etc.

References: