Bi-Directional Work-Sharing in Assembly Lines with Strict and Flexible Assembly Sequences

Yossi Bukchin¹, Tal Sofer

Department of Industrial Engineering
Faculty of engineering,
Tel Aviv University,
Tel-Aviv, 69978, ISRAEL

Submitted: October 2009

¹ Corresponding author
Bi–Directional Work-Sharing in Assembly Lines with Strict and Flexible Assembly Sequences

Abstract

Work-sharing between stations in assembly lines can reduce cycle time and improve throughput rate. This research addresses assembly lines in which tasks may be shared with their immediate upstream or downstream stations only. The study consists of three stages. In the first stage, we assume that all work content in the line can be shared. Then, the balanceability of the line, which is defined as its ability to achieve a perfect balance (identical workload in all workstations) via work-sharing, is examined. An algorithm and a linear programming (LP) formulation is proposed to provide the sharing proportion, namely, the percentage of the cycles each station performs the shared task. In the second stage, a line with partial sharing is investigated; conditions for balanceability are provided as well as a solution approach for obtaining the sharing proportions of the shared tasks. The last stage considers the most practical model in which sharing costs are involved, and the identity of the shared tasks is to be determined. Mixed-integer linear-programming (MILP) models and a custom-made search algorithm are developed to minimize the cycle time given a limited sharing budget, for strict and flexible assembly sequences. Experimentation to evaluate the performance of the formulations and the effect of the problem parameters on the run time is conducted.

1. Introduction and literature review

The primary phase of assembly line design is the assembly line balancing. In this phase, non-divisible assembly tasks are assigned to work-stations while trying to balance the workload among the stations as much as possible. Common objectives are minimizing the number of stations subject to a required throughput rate (denoted as SALBP-1, see Baybars 1986) or minimizing the cycle time (maximizing the throughput rate) subject to a given number of stations (denoted as SALBP-2). This combinatorial problem is NP-Hard (Wee and Magazine 1982), and numerous optimal and heuristic solution procedures have been presented during the last few decades (see literature reviews in Baybars 1986, Ghosh and Gagnon 1989 and Scholl and Becker 2006). Due to the nature of the problem, the balancing solution, either optimal or approximated, cannot guarantee a perfect balance, namely, an equal load in all stations. Hence, in most cases some stations will be more loaded than others, while the cycle time of the line will be determined by the most loaded station, namely, the bottleneck station.
In this paper, we suggest a new approach for improving throughput rate of non-perfectly balanced assembly lines via work-sharing. When applying work-sharing, the same task may be performed in different stations in different cycles. As a result, the workload division among the stations is improved, and the cycle time is reduced. The proposed approach is closely related to the cross-training literature, since the shared tasks must be performed by cross-trained workers due to the required overlapping in the workers’ capabilities.

Due to the increasing number of cross-training applications in modern industry, a wide literature on the subject can be found. Most of the research deals with the operational stage while some with the design. There are multiple ways to classify cross-training literature. One factor is the range of tasks that should be allocated to each worker. This range can start from a single task per worker (no overlapping, no work-sharing), through several tasks per worker (e.g. skill-chaining, see Hopp et al, 2004), up to workers who are capable of performing all tasks in the line. The latter is defined as full cross-training, and is discussed in Van Oyen et al. (2001) and Hopp et al. (2004). Another approach which relies on full cross-training is the well known bucket brigade, proposed by Bartholdi and Eisenstein (1996). Some of the literature addresses the zoning issue, namely, the range of work-stations allocated to each worker. “Cherry picking”, for example, (Hopp et al. 2004) suggests that one worker will assist all other stations; “2-skill Chaining” on the other hand, proposes work-sharing between adjacent stations, as each worker assists his/her neighbor. Some of the papers refer to the possibility of preemption. If preemption is not allowed, each worker must complete the task before handing it to another worker. In case of preemption, on the other hand, the worker can take over the task in the middle of its performance (McClain et al. 2000). Some research addresses operation policy of the line with work-sharing, namely, which worker will perform each shared task during the operation. Ostolaza et al. (1990) suggest focusing on the buffer size. Gel et al. (2007) set the principle of “fixed before shared”, and Anuar and Bukchin compare several operating rules, some are state dependent and other are state independent.

One may distinguish between moving worker and moving task policy in cross-training/work-sharing models. Most of the literature assumes the former, as the physical location of each task is fixed and the cross-trained worker moves from one location to another to assist his colleague. Such a policy is characterized by a less complex material handling system and lack of need for duplicating equipment. The worker, however, should be mobile, and the system may suffer from productivity loss due to this mobility. The latter policy, however, assumes that the physical location of performing a task may be changed dynamically. In this case the location of the worker is fixed; however, his work content may be changed. The approach proposed in this paper belongs to the moving task policy, and its advantages and disadvantages will be discussed later on.
This paper addresses the problem of applying work-sharing in working assembly lines, with an initial assignment of tasks to stations and given technological precedence constraints among tasks. The main objective of the paper is to analyze work-sharing when tasks can be performed by the adjacent upstream or downstream station. The conditions for balanceability of an assembly line, namely, its ability to be perfectly balanced via sharing, is first characterized. These conditions help to identify the shared tasks and the time proportion they should be performed in each station (denoted from now on as sharing proportion), in order to minimize cycle time. The second objective is to incorporate sharing costs, associated with training costs, tool duplication and increased system complexity. Consequently, we develop efficient solution approaches which determine the identity of the shared tasks to minimize the cycle time subject to a given budget for sharing. This research extends the study of Ostolaza et al. (1990) and Anuar and Bukchin (2006). The former were the first to present the term dynamic line balancing (DLB). This term refers to the operational side of work-sharing as "allowing tasks to be assigned 'on the fly' based on the current state of system". The basic idea is shifting additional capacity from low to high utilized stations by allowing different workers to perform the same task. This approach can be applied in existing lines to reduce idle time, increase flexibility, and reduce risk of injury caused by repetitive work. A dynamic control on the buffer inventory quantities between every two stations can be used to decide in which station the shared task will be done. Anuar and Bukchin (2006) suggested analytical conditions for line balanceability in lines where forward sharing is allowed (a task can be done by its current or the adjacent downstream station). They proposed several tools for the design and operation of such assembly lines, while assuming a strict order of assembly sequence. The proposed approach addresses the last two issues, as work-sharing in both directions is allowed and a flexible assembly sequence is considered.

The rest of the paper is organized as follows. The line with full sharing is investigated in Section 2. In this system we assume that any part of the workload can be shared, forward or backward, provided that work is transferred between adjacent stations only. Conditions for balanceability are developed as well as an algorithm and an LP model for calculating the sharing proportions in the balanceable and non-balanceable case, respectively. The situation of partial sharing, in which some of the work cannot be shared is addressed in Section 3. Conditions for balanceability as well as solution approaches for finding the balancing proportions in the balanceable and non-balanceable case are provided. In section 4, the cost associated with work-sharing is considered. Two mixed-integer linear-programming (MILP) models for minimizing the cycle time given a limited amount of sharing budget for the strict and flexible assembly sequence are presented. A custom-made algorithm is developed as well, which is based on the bottleneck segment principle. Last,
experimental results which evaluate the performance of the above solution approaches are given. Summary and concluding remarks are given in Section 5.

2. Line with full sharing

2.1. Problem description

Let $N$ denote the number of stations and $T_i$ the amount of workload assigned to station $i$. The ideal cycle time, obtained in a perfectly balanced line, is denoted by $T$, where $T = \frac{\sum_{i=1}^{N} T_i}{N}$. The cycle time of the line, $c$, is determined by the most loaded station, namely $c = \max(T_i)$. When sharing is applied, $p_i$ addresses the work division between station $i$ and station $(i+1)$, such that the percentage of work, originally assigned in station $i$ and remains in that station is denoted by $p_i$ and the work content transferred to station $(i+1)$ is denoted by $(1-p_i)$. Similarly, $q_i$ addresses the work division between station $i$ and station $(i-1)$. Recall that the shared task is not divisible, and division of workload between stations is done by setting the percentage of cycles the task (containing the whole station time in this section) will be performed in each station. Clearly, $0 \leq p_i \leq 1$ ($0 \leq q_i \leq 1$) and $p_i + q_i \geq 1$, to assure that the workload transferred from station $i$ does not exceed the total workload originally assigned to that station.

**Definition 1:** A line is balanceable if a perfect balance can be obtained via work-sharing between adjacent stations.

The above restriction will be noted from now on as the adjacency condition. Note that when work-sharing is applied, the station time is not constant and the average workload is considered.

2.2. Conditions for balanceability

**Theorem 1:** If all the assembly time can be shared in adjacent stations (full sharing), the conditions for balanceability are:

\[ T \leq T_1 + T_2 \]  \hspace{1cm} (1)

\[ \sum_{j=1}^{i-1} T_j \leq i - T \leq \sum_{j=1}^{i+1} T_j \]  \hspace{1cm} 2 \leq i \leq N - 1 \hspace{1cm} (2)

**Proof:** Let us use a sequential process starting with station 1. If station 1 is overloaded, namely, $T_1 - T > 0$, then a load of $(T_1 - T)$ must be transferred from station 1 to station 2 to obtain a load
of \( T \) in station 1. However, if \( T_1 - T > T \) (or \( T_1 > 2T \)), then some load has to be transferred from station 1 to station 3 (or a further station), and the adjacency condition does not hold. If station 1 is under-loaded, namely, \( T_i - T < 0 \), then a load of \( T - T_i \) should be transferred from station 2 back to station 1. However, if \( T - T_i > T_2 \) (or \( T > T_i + T_2 \)), then some load has to be transferred from station 3 (or a station further away) to station 1, contradicting the adjacency condition. In both cases, if \( T_i \leq 2T \) and \( T \leq T_1 + T_2 \), a load of \( T \) can be obtained in station 1 without violating the conditions of balanceability. Consequently, the load transferred from station 1 to station 2 results in a new workload in station 2 equal to \( T'_2 = T_2 + T_1 - T \). Similarly, when considering station 2, a load of \( [T'_2 - T] \) must be transferred, either from station 3 or to station 3, to reach a workload of \( T \) in station 2. We can see that if \( T'_2 - T > T \) (or \( T_2 + T_1 > 3T \)) or \( T - T'_2 > T_3 \) (or \( T_1 + T_2 + T_3 < 2T \)), the adjacency condition does not hold. On the other hand, if \( T_2 + T_1 \leq 3T \) and \( T_1 + T_2 + T_3 \geq 2T \), then a load of \( T \) can be obtained in station 2 without violating the adjacency condition. In general, for every station \( i \), if \( \sum_{j=1}^{i} T_j \leq (i + 1)T \) and \( \sum_{j=1}^{i+1} T_j \geq i \cdot T \) then a load of \( T \) can be obtained at that station without violating the adjacency condition. The union of all these conditions can be expressed as \( \sum_{j=1}^{i-1} T_j \leq i \cdot T \leq \sum_{j=1}^{i+1} T_j \), \( \forall i = 2..N - 1 \). If the above condition holds, a load of \( T \) can be obtained in every station \( j \) without violating the adjacency condition, and consequently, the line is balanceable. □

A sequential algorithm, based on the above condition, is proposed below for setting the sharing proportions to achieve a perfectly balanced line.

**Algorithm A1**

**Step 1:** Set \( i = 1 \) and \( T_i = T \).

**Step 2:** If \( i = N \), exit. The line is balanceable.

If \( T'_i > T \), transfer \( (T'_i - T) \) from station \( i \) to station \( i + 1 \).

If \( T'_i - T > T \), exit. The line is non-balanceable.

Update the time left in station \( i + 1 \): \( T'_{i+1} = \sum_{j=1}^{i+1} T_j - iT \).

Otherwise, if \( T'_i \leq T \), transfer \( (T - T'_i) \) from station \( i + 1 \) to station \( i \).

If \( T'_{i+1} < T - T'_i \), exit. The line is non-balanceable.
Update the time left in station $i+1$: $T'_{i+1} = \sum_{j=i}^{N} T_j - i\tau$

Step 3: Set $i = i + 1$, Go to step 2.

Note that the above model is somewhat non-realistic since (1) it assumes full sharing, and (2) the workload is not divided into tasks, namely, sharing can be done on any portion of the workload. However, if conditions (1) and (2) do not hold, then most practical constrained lines are non-balanceable as well, as will be discussed in the sequel.

### 2.3. Solution for non-balanceable lines

When the line is non-balanceable the cycle time and proportion of work-sharing can be found by the following linear programming (LP) formulation P1.

\[
\text{[P1] } \text{Lexmin} \left( c, - \left( \sum_{i=1}^{N-1} p_i + \sum_{i=2}^{N} q_i \right) \right) \tag{3}
\]

Subject to:

\[
c \geq p_i T_1 + (1 - q_i) T_2 \tag{4}
\]

\[
c \geq (p_i + q_i - 1) T_i + (1 - q_{i+1}) T_{i+1} + (1 - p_{i-1}) T_{i-1} \quad 2 \leq i \leq N - 1 \tag{5}
\]

\[
c \geq (1 - p_{N-1}) T_{N-1} + q_N T_N \tag{6}
\]

\[
0 \leq p_i \leq 1 \quad i = 1, \ldots, N - 1 \tag{7}
\]

\[
0 \leq q_i \leq 1 \quad i = 2, \ldots, N \tag{8}
\]

\[
(1 - p_i) + (1 - q_i) \leq 1 \quad i = 2, \ldots, N - 1 \tag{9}
\]

The objective function (3) minimizes the cycle time, $c$, as a primary objective and maximizes the component $\left( \sum_{i=1}^{N-1} p_i + \sum_{i=2}^{N} q_i \right)$ as a secondary objective. The purpose of the right hand component is to prevent unnecessary sharing, while acting as a tie breaker between different solutions with the same cycle time and different sharing proportions. Constraint (4) ensures that the cycle time is larger than or equal to the total workload of station 1, consisting of the original time remaining in station 1 after transferring some shared time to station 2, plus the time transferred from station 2 via sharing. Constraint set (5) addresses all stations but the first and the last one. It assures that the cycle time is larger than or equal to the original time remaining in the station plus the shared time transferred from the upstream and downstream stations. Similarly,
Constraint (6) refers to the last station. Constraint sets (7) and (8) ensures that the value of \( p_i \) and \( q_i \) is between zero and one. Constraint set (9) assures that the workload transferred to the upstream and downstream stations does not exceed the total work initially assigned to station \( j \).

Let us define two-way work-sharing as a situation in which station \( i \) shares time forward with station \( i+1 \), and at the same time, station \( i+1 \) shares time with station \( i \). One can easily show that a solution with two-way work-sharing can be replaced by an equivalent solution with one-way work-sharing. Hence, the optimal solution of P1 will never contain two-way work-sharing due to the tie break component in the objective function (3).

3. Line with fixed and shared time

In this section a more realistic situation is addressed, where only part of the workload in each station can be shared. Consequently, \( F_i \) denotes the fixed time of station \( i \), namely, time which cannot be shared. \( S^d_i \) (\( S^u_i \)) is the time which can be shared with the downstream (upstream) adjacent station. Clearly, \( S^u_i \equiv 0 \) and \( S^d_N \equiv 0 \) since station 1 (N) cannot share time with its upstream (downstream) station. In this subsection a strict assembly sequence is considered. Consequently, in each station \( i \) the time which can be shared backward, \( S^d_i \), is followed by the fixed time, \( F_i \), which is followed by the time which can be shared forward, \( S^u_i \). If, for example, tasks 2, 3 and 4 in Figure 1 are defined as fixed tasks, then task 1 can only be shared backward and tasks 5 and 6 can only be shared forward.

![Figure 1. Illustration of fixed and shared time](image)
3.1. Conditions for balanceable lines

Theorem 2: A line with fixed and shared time is balanceable if and only if the following conditions hold:

\[
\sum_{j=1}^{i} F_j + \sum_{j=2}^{i} S_j^d + \sum_{j=2}^{i} S_j^u \leq T \leq \sum_{j=1}^{i} F_j + \sum_{j=2}^{i} S_j^d + \sum_{j=2}^{i} S_j^u \quad i = 1, \ldots, N-1
\]

Proof: A perfectly balanced line can be achieved by only shifting workload of shared tasks between adjacent stations. Let us apply a sequential process starting with station 1. If station 1 is overloaded, namely, \( T_1 - T > 0 \), then a load of \((T_1 - T)\) must be transferred from station 1 to station 2 to obtain a load of \( T \) in station 1. The only work content that can be transferred is \( S_i^d \). If \( F_1 > T \), then some fixed load has to be transferred to station 2 and the adjacency condition does not hold. If station 1 is under-loaded, namely, \( T_1 - T < 0 \), then a load of \((T - T_1)\) should be transferred from station 2 back to station 1. The only work content that can be transferred in this case is \( S_i^u \). However, if \( F_1 + S_i^d + S_i^u < T \), then some load has to be transferred from the fixed task of station 2 to station 1, and the adjacency condition does not hold. In both cases, if \( F_1 \leq T \) and \( F_1 + S_1^d + S_2^u \geq T \), a load of \( T \) can be obtained in station 1 with time transfer from adjacent stations. Consequently, the load transfer from (to) station 2 results in a new workload in station 2, equal to \( T_2 = T_2 + T_1 - T \). Similarly, when considering station 2, a load of \( |T_2 - T| \) must be transferred, either from or to station 3 to reach a workload of \( T \) in station 2. We can see that if \( T_2 - T > S_2^d \) (or \( F_1 + S_1^d + S_2^u + F_2 > T \)) or \( T - T_2 > S_3^u \) (or \( F_1 + S_1^d + S_2^u + F_2 + S_2^d + S_3^u < T \)), the line is non-balanceable. On the other hand, if \( \frac{F_1 + S_1^d + S_2^u + F_2}{2} \leq T \) and \( \frac{F_1 + S_1^d + S_2^u + F_2 + S_2^d + S_3^u}{2} \geq T \), then a load of \( T \) can be obtained in station 2 without violating the conditions of balanceability and executing the fixed tasks in the station to which they are assigned to. In general, for every station \( i \), if \( \sum_{j=1}^{i} F_j + \sum_{j=1}^{i} S_j^d + \sum_{j=2}^{i} S_j^u \leq T \) and \( \sum_{j=1}^{i} F_j + \sum_{j=1}^{i} S_j^d + \sum_{j=2}^{i} S_j^u \geq T \), then a load of \( T \) can be obtained at this station without violating the adjacency condition and executing the fixed tasks in the station to which they are
assigned. The union of all these conditions can be expressed as expression (10). If the above condition holds, a load of $T$ can be obtained in every station $i$ without violating the adjacency condition and executing the fixed tasks in the station to which they are assigned to, and consequently, the line is balanceable. □

The rational of the above theorem is illustrated in Figure 2. In the upper graph, the original workload allocation to a 5-station assembly line without work-sharing is presented. A horizontal presentation is depicted in the lower graph. The vertical arrows mark the changes that should be made in the original line, via sharing, in order to get a perfect balance. For example, some load has to be transferred from station 2 to station 1 and station 3. One can see that the conditions for balanceability hold as each vertical arrow falls within the boundaries of the corresponding “time buffer”, which is equal to the sum of the shared tasks $S_i^d + S_{i+1}^u$ ($\forall i = 1,\ldots, N - 1$). Otherwise, the above condition is violated and load has to be transferred to non-adjacent stations to achieve balanceability.

![Figure 2. Illustration of the conditions for balanceability](image)

Setting the sharing proportions in order to achieve a perfectly balanced line, once Condition (10) holds, is calculated according to the following conditions, for each station $i$:

$$\sum_{j=1}^{i} F_j + \sum_{j=2}^{i} S_j^d + \sum_{j=2}^{i} S_j^u \geq T \quad \text{then} \quad p_i = \frac{i \cdot T - \left(\sum_{j=1}^{i} F_j + \sum_{j=1}^{i-1} S_j^d + \sum_{j=2}^{i} S_j^u\right)}{S_i^d} \quad \text{and} \quad q_{i+1} = 1.$$
\[
\sum_{j=1}^{i} F_j + \sum_{j=i}^{1} S_{j}^d + \sum_{j=2}^{1} S_{j}^u < T \quad \text{then} \quad q_{i+1} = \frac{\sum_{j=1}^{i} F_j + \sum_{j=1}^{i} S_{j}^d + \sum_{j=2}^{i} S_{j}^u - i \cdot T}{S_{i+1}^u} \quad \text{and} \quad p_i = 1.
\]

In the first case, the first \( i \) stations are overloaded in comparison to the first \( i \) stations in a perfectly balanced line. As a result, Task \( S_i^d \) has to be shared with the next downstream station (see, for example, station 2 in Figure 2). The shared task upstream from station \( i + 1 \) does not need to be shared in this case, so \( q_{i+1} = 1 \). In the second case, the first \( i \) stations are under-utilized in comparison to the first \( i \) stations in a perfectly balanced line. As a result, Task \( S_i^u \) has to be shared with the next upstream station (see, for example, station 1 in Figure 2). The shared task downstream from station \( i \) does not need to be shared in this case.

### 3.2. Solution for non-balanceable lines

When the line is non-balanceable the cycle time and proportion of work-sharing can be found by the following LP formulation P2.

\[
\begin{align*}
\text{[P2]} \quad & \text{Lexmin}(c, -(\sum_{i=1}^{N-1} p_i + \sum_{i=2}^{N} q_i)) \\
\text{Subject to} \quad & c \geq F_i + p_i S_i^d + (1-q_i) S_i^u \\
& c \geq F_i + (1-p_{i-1}) S_{i-1}^d + p_i S_i^d + q_i S_i^u + (1-q_i) S_{i+1}^u \quad 2 \leq i \leq N \\
& c \geq F_N + (1-p_{N-1}) S_{N-1}^d + q_N S_N^u \\
& 0 \leq p_i \leq 1 \quad i = 1, \ldots, N-1 \\
& 0 \leq q_i \leq 1 \quad i = 2, \ldots, N 
\end{align*}
\]

The objective function (11) is the same as in P1. The value of the cycle time is obtained by constraints sets (12-14), and constraint sets (15-16) set the value of the balancing proportions to be between zero and one.
4. Cycle time minimization subject to limited sharing costs

Thus far no downside of work-sharing was considered and hence the number of shared tasks could be as much as needed for minimizing the cycle time. Nevertheless, work-sharing means task replication, so that it can be done in two stations rather than one. In practice this leads to costs due to:

1. Duplication of equipment in all stations that perform the same task.
2. Cross-training of workers who perform the shared task.
3. Additional complexity associated with managing the line under work-sharing.

In the proposed model, we assume that each task requires its own unique tooling and equipment. Therefore, when sharing the task with another station the equipment has to be assigned in both stations. Moreover, in order to simplify the implementation, we let each task to be shared only between two stations. That is, if a task needs to be shared in a balanceable line with the previous station and next station, the line is non-balanceable.

 Clearly, the budget allocated to sharing is likely to be limited. Consequently, one would like to share only those tasks that have the most impact on improving the cycle time while exploiting to the utmost the given work-sharing budget. The sharing cost is assumed to be task dependent, namely, each task has its own sharing cost. However, the cost does not depend on the sharing proportion of the task. Nevertheless, if a task is transferred as a whole from one station to another, i.e., only one station eventually performs the task, the cost of sharing is zero. Note that the primary objective is to minimize the cycle time subject to the budget constraint, as discussed next. In case of a tie, the solution with the minimal total sharing cost will be selected. Hence, the problem is defined as finding an optimal solution that minimizes the cycle time as a primary objective and the total amount of sharing costs as a secondary (tie breaker) objective.

Since the proposed model is supposed to improve an existing operational assembly line, the initial assignment of tasks to stations is given, namely, the number of stations \( j \in 1, ..., J \) and the identity of the tasks in the line \( i \in 1, ..., I \) assigned to the stations.

Figure 3a and b illustrate the use of work-sharing for reducing the cycle time. The initial task assignment of 13 tasks in four stations is presented in Figure 1 where station 2 and 3 constitute the bottleneck that determines the cycle time \( c = 7 \) while stations 1 and 4 are idle part of the time. The optimal task assignment that minimizes the cycle time \( c = 5.875 \) via work-sharing is
presented in Figure 2 where tasks 4, 7, and 11 are shared. Note that tasks 8 and 12 were moved as a whole to the next stations, and are therefore not considered as shared. Recall that the graph presents the average workload of each station via sharing, while the shared tasks are not actually split between stations as it appears in the figure. In fact, each of the shared tasks is performed as a whole in different stations according to the sharing proportions given by the solution.

![Figure 3. Work-sharing illustration](image)

### 4.1. Model formulation for the strict assembly sequence

When sharing costs are considered, an optimal solution which minimizes the cycle time as a primary objective and the sharing costs as a secondary objective is considered. Let $S_i$ denote the station to which task $i$ is initially assigned, $i \in 1,\ldots,m$, $\sigma_j$ is the set of tasks initially assigned to station $j$, $j = 1,\ldots,N$, and $L_j$ ($F_j$) is the last (first) task performed in station $j$. Let $T_i$ and $C_i$ denote the performance time and sharing cost of task $i$, respectively. The total budget available for sharing is denoted by TAR. The following variables are needed for the MILP model, presented next. $ac_i$ is a decision variable which is equal to $C_i$ if task $i$ is shared, and 0 otherwise. Recall that if the task is transferred entirely to the adjacent station, no sharing costs are accounted. $x_i$ is equal to 1 if task $i$ is shared with the adjacent downstream station or moved as a whole to that station, and 0 otherwise. $y_i$ is equal to 1 if task $i$ is shared with the adjacent upstream station or moved as a whole to that station, and 0 otherwise. $z_i$ is equal to 1 if task $i$ is moved as a whole to the adjacent downstream station, and 0 otherwise. $w_i$ is equal to 1 if task $i$ is moved as a whole to the adjacent upstream station, and 0 otherwise.

The MILP formulation P3 is shown next.
\[ \text{[P3] Lexmin}(c, \sum_i ac_i) \] (17)

Subject to
\[
c \geq \sum_{i \in \sigma_{j,1}} (1 - p_i) T_j + \sum_{i \in \sigma_j} (p_i + q_i - 1) T_j + \sum_{i \in \sigma_{j,1}} (1 - q_i) T_j \quad j = 2..N-1
\] (18)

\[
c \geq \sum_{i \in \sigma_1} p_i T_i + \sum_{i \in \sigma_2} (1 - q_i) T_i
\] (19)

\[
c \geq \sum_{i \in \sigma_{N,1}} (1 - p_i) T_i + \sum_{i \in \sigma_N} q_i T_i
\] (20)

\[
1 - p_i \leq x_i \quad i = 1..m
\] (21)

\[
1 - q_i \leq y_i \quad i = 1..m
\] (22)

\[
1 - p_i \geq z_i \quad i = 1..m
\] (23)

\[
1 - q_i \geq w_i \quad i = 1..m
\] (24)

\[
p_{i+1} \leq 1 - x_i \quad \forall i, i+1 \in \sigma_j, \forall j \in J
\] (25)

\[
q_i \leq 1 - y_{i+1} \quad \forall i, i+1 \in \sigma_j, \forall j \in J
\] (26)

\[
x_{L_j} + y_{F_{j+1}} \leq 1 \quad \forall j = 1,..,N-1
\] (27)

\[
x_i + y_i \leq 1 \quad i = 1..m
\] (28)

\[
ac_i \geq C_i - (1 - x_i)M - z_iM \quad i = 1..m
\] (29)

\[
ac_i \geq C_i - (1 - y_i)M - w_iM \quad i = 1..m
\] (30)

\[
ac_i \geq 0 \quad i = 1..m
\] (31)

\[
TAR \geq \sum_i ac_i
\] (32)

\[
0 \leq p_i \leq 1 \quad \forall i \in j = 2,..,N
\] (33)

\[
0 \leq q_i \leq 1 \quad \forall i \in j = 2,..,N
\] (34)

\[
x_i \in \{0,1\}, \ y_i \in \{0,1\}, \ z_i \in \{0,1\}, \ w_i \in \{0,1\} \quad i = 1..m
\] (35)
The objective function (17) minimizes the cycle time as a primary objective and the sharing costs as a secondary objective in case of tie. Constraints (18-20), along with the objective function, determine the cycle time value. Constraint sets (21) and (22) ensure that the value of $x_i$ (or $y_i$) is zero only if task $i$ is neither shared partially nor transferred completely to station $j+1$ (or $j-1$). Constraint sets (23) and (24) ensure that $z_i$ (or $w_i$) is equal to zero if there is partial sharing or none of task $i$ with station $(j+1)$ (or $(j-1)$). Constraint set (25) enforces the strict order of tasks as it ensures that task $i$ in station $j$ can be shared with station $(j+1)$ only if task $(i+1)$ from the same station was transferred completely to station $(j+1)$. Similarly, constraint set (26) enforces the strict order of tasks in upstream sharing. Constraint set (27) prevents sharing of the last task in station $j$ (denoted as $E_{j+1}$) upstream along with sharing the first task in station $(j+1)$ (denoted as $E_{j-1}$) downstream. Constraint set (28) assures that each task is shared at most with one additional station. Constraint sets (29-31) count the sharing cost of task $i$ only in case $0 < p_i < 1$ or $0 < q_i < 1$. Finally, constraint set (32) ensures that sharing costs will not exceed the budget limit. Constraint sets (33-35) are non-negativity and integrality constraints.

The solution of P3 may belong to one of the next cases:

1. The line is balanceable within the given budget.
2. The line is non-balanceable due to one of the following:
   2.1 Non-feasible task assignment due to the restriction to share tasks between adjacent stations.
   2.2 Budget limitations. The existing budget does not allow sharing of all tasks that should be shared in order to reach a balanceable line.

4.2. Model formulation for the flexible assembly sequence

The relaxation of the strict order constraint results in more sharing options, which enlarges the feasible region. By modifying formulation P3 to capture the new model with precedence constraints, formulation P4 is obtained. Clearly, the difference between the two formulations is in the precedence constraints. Note that we still keep a strict order between stations, namely, the tasks that were originally assigned to some station $j$ should be completed before starting the performance of the tasks in station $j+1$ (or in other words, switching tasks of different stations is not allowed). The reason for this is our purpose to apply the new approach on an operating line, and therefore, minimizing the changes of the existing line.
Let us define \( P_{(k)} \) as the set of immediate precedence of task \( k \) assigned initially to the same station. The new formulation P4 is presented next.

\[
[P4] \quad \text{Lexmin}(c_i \sum_j ac_j)
\]

Subject to:

\[
(18)-(24), (27)-(35)
\]

\[
p_k \leq 1 - x_i \quad \forall i \in P_{(k)}
\]

\[
q_i \leq 1 - y_k \quad \forall i \in P_{(k)}
\]

\[
y_i \leq p_i \quad \forall i \in j+1, \forall l \in j, \forall j = 1, ..., J-1
\]

\[
x_i \leq q_i \quad \forall i \in j+1, \forall l \in j, \forall j = 1, ..., J-1
\]

The objective function and constraint sets (18)-(24) and (27)-(35) remain the same as in P3 (the strict assembly order case). Constraint set (37) keeps the precedence order of tasks while sharing a task from station \( j \) with station \( (j+1) \). It ensures that task \( i \) in station \( j \), which is an immediate precedence to task \( k \) also located in station \( j \), can be shared with station \( (j+1) \) only if task \( k \) from the same station was previously transferred as a whole to station \( (j+1) \). The same applies to the upstream sharing captured by constraint set (38). Constraint sets (39) and (40) maintain the strict order between stations, so that all of station \( j \)'s tasks will be completed before starting the tasks of station \( (j+1) \). That is, if there is a task from station \( (j+1) \) that is shared with station \( j \) or moved as a whole to that station (so that \( y_i = 1 \)), then no task from station \( j \) will be shared with station \( (j+1) \) or moved to that station. Otherwise the possibility of switching tasks between stations \( j \) and \( (j+1) \) exists, which contradicts the assumption of strict order between stations.

4.3. Bottleneck Segment Algorithm (Algorithm BSA)

The above formulations can be applied for solving the work-sharing problem with either strict or flexible assembly sequence using commercial software. Still, in this section we suggest a custom-made search algorithm, to be used as an alternative to the direct solution of the MILP formulation. The search algorithm is based on gradual expenditure of the sharing budget while identifying the bottleneck station(s) in the line in each iteration of the algorithm. We first suggest an algorithm for the strict assembly sequence case and then modify it to the case of flexible assembly sequence.
Definition 2: Station \( j \) is defined as a Bottleneck (BN) station if

\[
j = \arg \max_i \left\{ \sum_{i \in r_j} [(1 - p_i)T_i^+ + \sum_{i \in r_j} (p_i + q_i - 1)T_i^- + \sum_{i \in r_j} (1 - q_i)T_i^-], \text{ given the sharing proportions.} \right\}
\]

Definition 3: A set of consecutive stations is defined as a Bottleneck Segment (BNS) if:
1. Each station in the set is a BN.
2. Each consecutive pair of stations in the set shares time (either upstream or downstream).
3. The first station in the set does not share time with the adjacent upstream station (if exists) and the last station does not share time with the adjacent downstream station (if exists).

A BNS example, consisting of stations 2 and 3 is presented in Figure 4. One can see that (1) each of the stations is a BN; (2) the two stations share task 8; and (3) station 2 does not share any time with station 1 and station 3 does not share any time with station 4.

Recall that two types of variables are involved here, the balancing proportions and the identity of the shared tasks. Given the identity of the shared tasks, the problem can be easily solved via P2 which minimizes the cycle time. Hence, the proposed algorithm is based on tree generation, where each node in the tree is a feasible solution of the problem. The root of the search tree contains a feasible solution where all tasks are fixed and no sharing exists. In each node down the tree an additional fixed task becomes shared, the cumulative sharing cost is calculated and the new cycle time is obtained using P2. If the cumulative sharing cost exceeds the given budget, the cycle time of that node is set to infinity. Naturally, the effectiveness of the algorithm depends on the sequence of the tree generation. This is done based on identifying the BNS at each stage and sharing both the first and the last task in the BNS, resulting in two new descendant nodes. This way the cycle time is decreased by transferring workload from the BNS to adjacent, less loaded, stations. Once there is more than one BNS in the line, all BNSs need to be handled in order to reduce the cycle time. Hence, the algorithm goes over all BNSs in a lexicographic order. The flowchart of the algorithm is presented in Figure 5, and the main stages are summarized as follows:

Step 1: Create root node – all tasks are fixed and no sharing exists.
Step 2: Choose the node with the minimal cycle time to be the parent node. Identify single or multiple BNS in the parent node.

Step 3: Create descendants (if possible) by sharing the first task (if station 1 is not included in the BNS) and the last task (if station N is not included in the BNS) in each BNS.

Step 4: Eliminate duplications – in case more than one node contains the same set of shared tasks, keep one instance only.

Step 5: Calculate cycle time and sharing cost using P2. If the sharing cost exceeds the given budget, the node is infeasible and the cycle time is set to infinity.

Step 6: Continue with 2-5 until reaching optimality.

Figure 5. Flowchart of algorithm BNS

Theorem 3: Algorithm BSA provides an optimal solution to P3.

Proof: P2 provides an optimal solution for a given configuration with a given identity of the fixed and shared tasks. The optimal solution of P2 will always contain at least one BNS (otherwise, the cycle time could have been improved without changing the identity of the shared tasks). Consequently, the optimal cycle time obtained by P2 can be decreased only by transferring workload from BNS stations to non-BNS stations. Due to the strict assembly sequence constraint the workload can be transferred either by sharing the first task in the BNS with the upstream station or by sharing the last task in the BNS with the downstream station. Since the above procedure is the only way to improve the cycle time given the identity of the fixed and shared
tasks, starting with a root node with no sharing and developing all relevant descendants would necessarily yield an optimal solution. □

When the strict assembly sequence constraint is partially relaxed, we face the work-sharing problem with flexible assembly sequence. The sequence flexibility is then expressed by precedence relationships among tasks. Algorithm BSA can be easily modified to address this case since the only change is associated with the number of descendents of each parent node. As in the algorithm for the strict assembly sequence there were at most two descendents for each node, this time this number may increase drastically, depending on the structure of the precedence diagram. Since we still develop the whole search tree, optimality is maintained, and the proof is similar to the one given in Theorem 3.

The structure of the solution obtained by Algorithm BSA will correspond to one of the cases the follows formulation P3, at the bottom of subsection 4.1. Another property of the optimal solution is discussed next.

**Proposition 2:** There is an optimal solution of BSA and P3 or P4 in which there is at most one shared task between every two consecutive stations; if $a_{ij} > 0 \forall i$, then it holds for any optimal solution.

**Proof:** Assume there are two tasks, $l$ and $k$ with task duration $t_l$ and $t_k$, respectively. Assume that the two tasks, originally belong to station $j$, share time with station $(j+1)$ in the optimal solution. Consequently, $0 < p_l < 1$, $0 < p_k < 1$, resulting in a sharing cost of $a_{il} + a_{ik}$. Let $S_j = (1 - p_l)t_l + (1 - p_k)t_k$ be the total time transferred from station $j$ to station $(j+1)$. Clearly, $S_j = t_l + t_k$.

Next, we show that the sharing cost can always be decreased by transferring the same amount of time when sharing only one of these tasks. Let $p'_l$ and $p'_k$ be the new sharing proportions for tasks $l$ and $k$, respectively, as at least one of these values is equal to zero or one (consuming no sharing costs).

Consider the following cases:

- $S_j > t_l$. Task $l$ is then transferred completely to station $j+1$, namely, $p'_l = 0$ and

$$ (1 - p'_l) = \frac{S_j - t_l}{t_k}, \text{ or } p'_l = 1 - \frac{S_j - t_l}{t_k}. $$

- $S_j < t_l$. Task $k$ is then performed completely in station $j$, namely, $p'_k = 1$ and

$$ (1 - p'_k) = \frac{S_j}{t_l}, \text{ or } p'_k = 1 - \frac{S_j}{t_l}. $$
• $S_j = t_j$. Task $l$ is then transferred completely to station $j+1$, namely, $p'_l = 0$ and Task $k$ is then done completely in station $j$, namely, $p'_k = 1$.

Since any pair or shared tasks can be reduced to a single shared task, we can decrease any number of shared tasks into a single one by using an iterative procedure based on the above cases. □

4.4. Experimentation

Two sets of experiments were conducted, for the strict and flexible assembly sequence. The purpose of the experiments was to compare the performance of the MILP formulation (P3 or P4 solved by commercial package OPL-Studio© with CPLEX engine) and the BSA algorithm for different values of problem parameters. Similar experimentation was designed for the two problems, with two differences: (1) larger instances (expressed by the number of stations) were solved for the strict order problems, due to the smaller feasible region of this problem; (2) an additional factor was added in the flexible sequence case which examines a different level of sequence flexibility. Hence, the following factors were considered, as factors/levels denoted by $s$ and $f$ are considered for the strict and flexible sequence only, respectively:

1. A – Line length with three/two levels:
   - A1 - Short line – $7^s, 5^f$ stations.
   - A2 - Medium line - $15^s, 10^f$ stations.
   - A3 - Long line – $30^s$ stations.

2. B – Average number of tasks per station in initial assignment of the line with two levels:
   - B1 - Low average number of tasks per station - $4^s, 3^f$.
   - B2 - High average number of tasks per station - $8^s, 6^f$.

3. C – Task time variability with two levels:
   - C1 - Low variance – the task time is distributed uniformly [3, 5].
   - C2 - High variance – the task time is distributed uniformly [2, 8].

4. D – Sharing cost variability with two levels:
   - D1 - Low variance – sharing cost of each task is distributed uniformly [5, 9].
   - D2 - High variance – sharing cost of each task is distributed uniformly [5, 15].

5. E – The total available budget (TAB) for improving the cycle time through DLB with two levels:
   - E1 - High level of budget available - 40% of the sum of all sharing costs of all tasks in line.
   - E2 - Low level of budget available - 7% of the sum of all sharing costs of all tasks in line.
6. **F** – The value of the F-ratio (see Dar-El, 1973) in each station with two levels:
   - **F1** – High level (F-ratio above 0.6).
   - **F2** – Low level (F-ratio below 0.4).

Note that factors A to E were examined for both problems. Factor F, which was added for the flexible sequence problem, uses the F-ratio performance measure, which indicates the flexibility of the assembly sequence in each station. Consider a precedence matrix in which each element \( I_{lm} \) is equal to 1 if Task \( l \) precedes Task \( m \) and zero otherwise. The F-ratio of a station is, accordingly, calculated as \( \frac{2Z}{n(n-1)} \) when \( n \) is the number of tasks in the station and \( Z \) is the number of zeroes in the precedence matrix. When the F-ratio is equal to zero, the sequence of the tasks cannot be changed, i.e. there is no flexibility in the assembly sequence. A value of one, on the other hand, indicates no precedence constraints among the tasks, resulting in maximal flexibility.

Four replications were generated for the strict order problems and three replications for the flexible sequence problems, resulting in a total of 192 experiments for each problem.

In general, as could be expected, longer run time was required in both the MILP and the BSA algorithm for solving the problems of the flexible assembly sequence, due to the larger feasible region. Hence, the smaller problems (smaller number of stations) were solved for the flexible assembly sequence, following a preliminary experimentation. Consequently, all problems in the experiments were solved to optimum, and the main purpose of the experimentation was to compare the two solution approaches. To this end, each instance was solved by MILP and Algorithm BSA.

ANOVA was performed to examine the effect of the factors on the run time (significance level of 0.95). Results show that both for BSA and MILP the run time increases with the number of stations, the number of tasks per stations, and the level of available budget. These effects are quite intuitive since the run time increases with the problem size (number of stations) and the size of the feasible region, derived from the number of tasks per station and the available budget. In addition, we have found only in algorithm BSA that the run time of Algorithm BSA decreases with the variance of the task time and the sharing cost, possibly since large variance of these factors may cause fathoming of many more branches. Finally, for the case of flexible assembly sequence, the run time increases with the F-ratio in both Algorithm BSA and MILP. This result was quite expected, since high F-ratio implies much more sharing options.

In the next stage of the experiments we have compared the performance of Algorithm BSA with the MILP solution of P3 in the strict assembly sequence case or P4 in the flexible assembly sequence case. In order to eliminate the influence of the computer and coding efficiency, the
number of created nodes was used as a performance measure of the performance of the solution approach. Algorithm BSA as well as MILP was solved for each instance and an ANOVA table was conducted to examine the significance of the comparison, with a dependent variable equal to 1 if BSA outperformed MILP and (-1) otherwise. The model was found significant, and the results associated with the significant main effects and interactions are presented in Table 1.

Table 1. Summary of results – BSA versus MILP

<table>
<thead>
<tr>
<th>Factor/Interaction</th>
<th>BSA&lt;MILP</th>
<th>BSA&gt;MILP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Low budget (E2)</td>
<td>85%</td>
<td>15%</td>
</tr>
<tr>
<td>High budget (E1)</td>
<td>28%</td>
<td>72%</td>
</tr>
<tr>
<td>7 stations (A1)</td>
<td>86%</td>
<td>14%</td>
</tr>
<tr>
<td>15 stations (A2)</td>
<td>45%</td>
<td>55%</td>
</tr>
<tr>
<td>30 stations (A3)</td>
<td>39%</td>
<td>61%</td>
</tr>
<tr>
<td>Time variability – U[3,5] (C1)</td>
<td>68%</td>
<td>32%</td>
</tr>
<tr>
<td>Time variability – U[2,8] (C2)</td>
<td>46%</td>
<td>54%</td>
</tr>
<tr>
<td>Low budget and 15 stations (A1 and E2)</td>
<td>100%</td>
<td>0%</td>
</tr>
<tr>
<td>High budget and 30 stations (A3 and E2)</td>
<td>75%</td>
<td>25%</td>
</tr>
<tr>
<td>High budget and 30 stations (A3 and E1)</td>
<td>0%</td>
<td>100%</td>
</tr>
<tr>
<td>High F-ratio (F1)</td>
<td>23%</td>
<td>77%</td>
</tr>
<tr>
<td>Low F-ratio (F2)</td>
<td>48%</td>
<td>52%</td>
</tr>
<tr>
<td>Low budget (E2)</td>
<td>58%</td>
<td>42%</td>
</tr>
<tr>
<td>High budget (E1)</td>
<td>13%</td>
<td>87%</td>
</tr>
<tr>
<td>10 stations (A2)</td>
<td>42%</td>
<td>58%</td>
</tr>
<tr>
<td>Low budget and 10 stations (E2 and A2)</td>
<td>67%</td>
<td>33%</td>
</tr>
<tr>
<td>Low budget and low F-ratio (E2 and F2)</td>
<td>75%</td>
<td>25%</td>
</tr>
<tr>
<td>Low budget and high F-ratio (E2 and F1)</td>
<td>38%</td>
<td>62%</td>
</tr>
<tr>
<td>Low budget, low F-ratio and 5 station (E2, F2 and A1)</td>
<td>96%</td>
<td>4%</td>
</tr>
</tbody>
</table>

Based on the results in Table 1, one can conclude the following:

- Algorithm BSA performs much better than MILP when the available budget for sharing is relatively low, and vice versa.
- Algorithm BSA performs relatively well for a small number of stations, compared to MILP. This trend is identified both in the strict and flexible assembly sequence; still, in the former Algorithm BSA outperforms MILP in the small number of stations, while in the latter MILP outperforms Algorithm BSA in both cases.
• Algorithm BSA performs well in a smaller time variability compared to MILP. This effect is insignificant for flexible assembly sequence case.

• The best parameter combination for Algorithm BSA, compared with, is short lines and low sharing budget. In this case Algorithm BSA provided better results than MILP in 100% of the problems in the strict assembly sequence case and in 67% of the cases in the flexible assembly sequence.

• In the flexible assembly sequence case, the F-ratio significantly affects the relative performance of Algorithm BSA versus MILP. One can see that Algorithm BSA outperforms MILP in 77% of the problems for low F-ratio and in only 48% of the problems in high F-ratio. Consequently, the best parameter combination for the flexible assembly sequence consists of low sharing budget, short line and low F-ratio. Under these conditions, Algorithm BSA outperforms MILP in 96% of the problems.

The above analysis suggests that the available budget is a critical factor that indicates when each procedure should be used. Therefore, other experiments were conducted to examine the effect of this factor in a higher resolution. Fifteen new problems were generated, and the parameters other than the available sharing costs were randomly generated for each problem in typical ranges derived from the previous experiments. Thirteen levels of available budgets were examined, starting with 5% of the total sharing cost up to 29% of the total sharing cost, with increments of 2%. The number of combinations was $15 \cdot 13 = 195$ experiments.

Results shown in Figure 6 strengthen the above observation in which Algorithm BSA performs relatively well compared with MILP for a small amount of budget. This result is quite intuitive, since Algorithm BSA starts with a system with no sharing and gradually adds sharing costs when needed. Naturally, when the available budget is low, the algorithm will end relatively fast. In particular, results show that BSA outperforms MILP in all cases when the level of budgets for work-sharing is 5% of the total sharing cost. BSA performance deteriorated as the budget increases. Starting from 17% of the total sharing cost and up, a steady state performance is reached for both BSA and MILP. This can be explained by the fact that the budget was sufficient to reach a balanceable line in most cases.
5. Concluding remarks

In this study, a new approach for increasing the throughput rate of an existing operational assembly line via work-sharing among stations was developed. The research addressed bi-directional work-sharing between stations as well as flexible assembly sequence of tasks. We have first identified conditions for balanceability (the ability to reach a perfect balance of work among stations via sharing) of an assembly line where all work content could be shared, provided that sharing occurs between adjacent stations only. An LP formulation was developed to find the sharing proportions for non-balanceable lines. Next, the case of partial work-sharing has been addressed, where the identity of the shared and fixed work content was given. Tools for finding the sharing proportions were developed for both the balanceable and non-balanceable cases. Last, the tradeoff between sharing costs and throughput rate has been investigated. To this end, two MILP formulations were developed, for minimizing the cycle time via work-sharing when the sharing cost of each task is given and the budget for work-sharing is limited. Both formulations provide both the identity of the shared tasks and the sharing proportions for each task, when addressing both strict and flexible assembly sequence of tasks. In addition, a custom-made algorithm, called Algorithm BSA, which is based on the relaxing bottleneck segments of the line, has been suggested. Experiments showed that relatively large-scale problems can be solved in a reasonable time. Moreover, performance comparison between the two solution approaches have shown that
Algorithm BSA outperforms direct solution of the MILP formulation for parameter combinations of short lines, low available budget and low flexibility of the assembly sequence.

References