Original Research

Tackling Demand Fluctuation through Novel Flexible Assembly Line Balancing Algorithm

Ari Setiawan[1*], Christopher Yehuda[2], Eka Kurnia Asih Pakpahan[3]

Received: 2 August 2025 / Accepted: 20 September 2025 / Published online: 21 September 2025

*** Corresponding Author Email, ari_setiawan@ithb.ac.id**

1,2,3- Department of Industrial Engineering, Harapan Bangsa Institute of Technology, Bandung, Indonesia

Abstract
This paper studied the situation faced by an upstream textile manufacturer company supplying seat cover fabrics to major automotive producers which need to dynamically adjust task distribution on their assembly line in response to demand fluctuations, while improving load balance among workstations. The company itself has implemented lean manufacturing concept, specifically the takt-time rules to organize the pace of its assembly line. Problems occur when demand increases significantly and one (or several) workstations in the assembly line suffers capacity insufficiency. To rectify the insufficiency, parallel processing becomes necessary. The company relies heavily on manual labor and has limited number of workers; therefore, parallel processing can only be implemented by utilizing the available workers. To do this, a new algorithm was designed. The algorithm works to identify which workstations suffer insufficiency, which workstations are available to perform parallel processing and how long the parallel processing should be performed. The algorithm is numerically tested. Four cases which cover various situations commonly found in real world assembly line are designed as test cases. The experiment showed that the new algorithm managed to reallocate tasks among workstations in such a way that the targeted takt-time was achieved while the line smoothness index was improved.

Keywords – Assembly line; Flexibility; Line balancing; Takt-time; Smoothness index.

INTRODUCTION

Modern manufacturing enterprises are confronted with increasingly complex challenges due to heightened global competition. To achieve sustainable competitiveness, organizations are compelled to enhance both operational efficiency and overall productivity. Market-driven demands for superior product quality, rapid delivery, and cost-effectiveness necessitate optimal performance across all production lines. In such environment, it becomes imperative for firms not only to sustain their production capacity but also to meticulously manage manufacturing processes to ensure operational efficiency and adaptability in the face of fluctuating customer demand [1].

A manufacturing company producing seat covers for major automotive brands is currently facing the challenge of highly fluctuating customer demand. Although the company has so far been able to fulfill all customer orders, this achievement has come at the expense of operational efficiency. The production line is experiencing performance disruptions due to uneven workload distribution, indicating a significant imbalance in the system. These reactive and inefficient practices pose a serious threat to the company's long-term sustainability and highlight the urgent need for a more agile, adaptive, and resilient production management strategy, particularly in the management of its assembly line. The company itself is in the early stage of lean manufacturing implementation. The lean manufacturing itself introduced the concept of takt-time as a pacing mechanism to synchronize production with customer demand [2][3]. The core idea is to adjust the speed of the assembly line so that it

consistently aligns with the rate of demand. When demand varies, the takt-time must be dynamically recalculated followed by task reallocation strategies among workstations to ensure that the new takt-time is consistently met. Task reallocation may involve merging tasks from multiple workstations into a single station or, conversely, splitting tasks originally assigned to a single workstation across several stations. However, not all tasks can be divided into smaller sub-tasks. In such cases, parallel processing becomes necessary to shorten the cycle-time. This paper addresses the task reallocation problem for the company's assembly line such that the fluctuating demand is always fulfilled and balance workload between workstations is achieved [4].

Assembly Line Balancing (ALB) problem is a classical issue in operations research and industrial engineering, which aims to optimally assign tasks to workstations. The assignment must comply with specific constraints, such as precedence relations and cycle time limits. The typical objectives include minimizing the number of workstations or balancing workloads across stations to enhance overall efficiency[5]. Most ALB problem deals with deterministic demand and fixed task times which limit their applicability in dynamic environments. Recognizing this limit, Flexible Assembly Systems (FAS) emerges as a strategic solution to address the challenges of assembly line management, particularly in manufacturing environments characterized by highly variable demand, diverse product types (low-volume, high-mix production), and unstable workload distribution across stations. FAS enables dynamic task assignments, rapid reconfiguration of workstations, and adaptive responses to changing production requirements. These capabilities are crucial for maintaining balance and efficiency in modern manufacturing systems that are increasingly driven by customer demand [6].

Parallel processing in assembly lines has become known as strategy to shorten cycle time [4] especially when tasks cannot be further divided into smaller sub-tasks. Its implementation introduces a critical requirement: the availability of additional resources. The availability of additional resources enables the duplication of workstations (or more), thus shortening the cycle time at a particular workstation Nevertheless, not every manufacturing company has additional resources readily available in their system. In such system, the only way that parallel processing could be performed is by utilizing existing operators who experience idle time in other parts of the assembly line.

This paper specifically addresses the problem of tasks reallocation where demand varies, and parallel processing is taken as step for shortening the workstation's cycle time. The parallel processing itself is performed by utilizing idle time operators in other parts of the assembly line due to the limited number of resources. It raises important operational questions: *which workstation on the assembly line has insufficient capacity to match the takt-time? Which workstation should be utilized to help the insufficient? At what point should a workstation begin its task helping the insufficient? How many units of workload should be transferred from the insufficient workstation to the helpers?* This paper contributes to answering these questions by proposing a novel workload reallocation algorithm designed to create flexible and balance assembly line configurations. It is important particularly in labor-intensive environments where demand fluctuations and human resource constraints are prevalent. The algorithm is then tested into various cases to analyze its applicability.

The paper will be divided into five sections. The introduction of the problem is in the first section. The second section discusses the literature review and position of this paper. The third section discusses the research method, algorithm development and implementation, the fourth section discusses the result, and the fifth section gives conclusion and recommendation.

LITERATURE REVIEW

Assembly lines are widely employed in manufacturing to produce standardized goods in large volumes. In this system, semi completed products move through a series of workstations, where each station carries out a specific set of tasks contributing to the product's assembly. This sequential process continues until the product is fully assembled and exits the line. The Assembly Line Balancing (ALB) problem represents a diverse group of optimization challenges focused on determining the best way to distribute tasks among workstations along the assembly line. In many cases, improvement in assembly line significantly increased production line productivity and capacity [7], [8], [9].

Among the various formulations of the Assembly Line Balancing (ALB) problem, the Simple Assembly Line Balancing (SALB) problem is the most extensively examined in literature. SALB represents a fundamental and widely adopted model within the broader family of line balancing problems. This problem addresses the assembly of a single product whose manufacturing process is decomposed into a set of basic tasks with known and deterministic processing times. These tasks must be assigned to several serially arranged workstations, which operate under a similar cycle time. A valid solution to the SALB problem involves assigning tasks to workstations such that the total processing time of tasks at any station does not exceed the cycle time, and all precedent constraints are maintained. The primary objective of the SALB problem is to optimize line performance, typically by minimizing the total idle time. There are several problem variants for SALB. The first is SALB-

1, which seeks to minimize the number of workstations for a given cycle time. The second is SALB-2, which aims to minimize the cycle time for a fixed number of workstations. The third is SALB-E, which simultaneously optimizes both the number of workstations and the cycle time [4].

SALB focuses on static and deterministic conditions. Real-world environments are far more dynamic. Key challenges include task time variability, worker skill heterogeneity, stochastic demand, task incompatibility, and limited resources. Research by [10] examined dynamic takt time decision on an assembly line to accommodate task time variability; [11] developed framework which considers processing time variability and product defect rate into line balancing process, which provides basic knowledge to deal with uncertainty; and [12] proposed the use of two different cycle time to anticipate demand fluctuations. On the other hand, worker skill heterogeneity was considered by [13] and [14]. Research by [13] modeled the effect of worker's skill towards processing time while [14] modeled the effect of worker's multi-skilled capability towards task assignment flexibility. Other research introduces extended models like U-shaped lines [15], mixed-model lines (MALB) [16], and two-sided lines (TSALB) [17] as ways to allocate a multi skilled worker to a number of machines.

In response to variable demand, reallocation of tasks becomes essential to maintain line balance. Literature reveals three main strategies to deal with this. The first is task consolidation, which means combining tasks from multiple stations to reduce idle time when demand drops. The second is task splitting, which means dividing tasks into multiple stations or operators when demand increases or cycle-time decreases. The third is parallel processing, which is duplicating resources to shorten the workstation's cycle time. The work of [4] designed heuristic algorithm to find the best task allocation strategy for a variant of SALB which allows parallel processing. They assume that the required resources to enable parallel processing is always available. While literature provides a solid theoretical foundation, most studies focus on idealized conditions. Few papers explore human-centered reallocation, especially under constraints of limited workforce and manual operations. Decision support tools for real-time task reassignment in response to demand shifts are underdeveloped. There is a pressing need for practical, algorithm-based frameworks that enable flexible task redistribution, operator reallocation based on workload and idle times, parallel execution of tasks using existing human resources. This study contributes to this space by developing a novel algorithm to dynamically reassign tasks and optimize the balance of manual assembly lines under fluctuating demand scenarios.

RESEARCH METHOD

A common challenge in assembly line operations can be described as follows: given a specified production demand and the available working hours, the takt-time is calculated to determine the pace at which products must be completed to meet demand. Subsequently, the cycle-time at each workstation along the assembly line is evaluated to ensure that it remains below the takt-time. If all workstation cycle-times are within this limit, the assembly line is considered capable of fulfilling the demand. However, if even a single workstation has a cycle-time that exceeds the takt-time, the line will be unable to meet the production target within the production time frame.

In response, lean manufacturing principles suggest for a detailed examination of each operation within the workstations and recommend the elimination of non-value-added activities to streamline processes, thereby reducing cycle-times to meet the takt-time. If this streamlining process proves insufficient and the takt-time is still not achieved, further intervention may involve decomposing complex operations into smaller sub-operations, each with a shorter execution time. These sub-operations can then be reassigned to new workstations, ensuring that the whole line is operating with a under the takt-time. This latter strategy, however, depends upon the fulfillment of two critical preconditions: (1) the original operations must be able to be divided into sub-operations, and (2) additional labor resources must be available to be assigned to the newly established workstations. In many practical scenarios, these conditions cannot be met. *This study addresses assembly line balancing in contexts where such prerequisites are absent and proposes a practical algorithm for reallocating tasks when workstations are identified with cycle-times exceeding the takt-time.* The objective of the algorithm is two folds, the first is fulfilling the targeted takt time and the second is balancing workload between workstations. The algorithm works with several constraints following the real system where this problem was originally recognized: each operation is bound to a sequential rule (precedence) and it cannot be divided into smaller sub-operations; the working hours and the number of operators currently in the system cannot be increased because the company urges full utilization of current operational time and resources.

On the following sub-sections, the algorithm development and implementation will be explained.

I. Algorithm Development

The proposed algorithm was developed for reallocating tasks among workstations on a particular assembly line. The assembly line produces a certain product with daily demand D units and H working hours. The assembly process is consisted of O assembly operations, where each is identified by the index of o ($o = 1,2,\dots,O$). Each operation cannot be divided into smaller sub-operations. There are N workstations, and each workstation is identified by the index of n, where $n = 1,2,\dots,N$. Each operation is assigned to a workstation, which means that $O = N$. Daily demand (D) varies over time, which creates challenges for the production planner in adjusting tasks allocation among workstations. The proposed algorithm is aimed at helping the planning process. It is divided into three sub algorithms. Sub-algorithm 1 aims at evaluating the ability of each workstation on fulfilling demand and determining which mechanism should be pursued for conducting tasks reallocation (see sub-algorithm 1 on Table I); Sub-algorithm 2 perform tasks reallocation through idle time adjustment, and Sub-algorithm 3 perform the same thing through total idle and waiting time adjustment (see sub-algorithm 3, on Table III).

TABLE I.
SUB ALGORITHM 1 (GENERATING WORKSTATIONS PROFILE)

Input	:	Number of operations ($\boldsymbol{O}$), processing time for each operation ($\boldsymbol{t_o}$), number of workstations ($\boldsymbol{N}$), assignment of operations to each workstation ($\boldsymbol{X_{o,n}}$), demand ($\boldsymbol{D}$) and operational time ($\boldsymbol{H}$).
Step 1	:	Calculate takt-time (T) for fulfilling demand by dividing demand by the operational time; $T = \frac{D}{H}$.
Step 2	:	Evaluate the ability of each workstation to fulfill demand.
2a	:	Calculate the cycle time per unit (c_n) for all operations assigned to workstation n; $c_n = \sum_{o=1}^{O} X_{o,n} t_o$.
2b	:	Determine the start time for the first unit at each workstation ($s_n, n = 1,2,\dots,N$), which is equal to the completion time for the first unit at the previous workstation $(n-1)$ or equals to 0, for $n = 1$; $s_n = \begin{cases} 0, & if\ n = 1 \\ f_{n-1}, & if\ n > 1 \end{cases}$; Completion time for the first unit at workstation n is the summation of start time for the first unit at workstation n and cycle time per unit at workstation-n; $f_n = s_n + c_n$.
2c	:	Total cycle time for workstation n (C_n) by multiplying demand and the cycle time per unit at workstation-n; $C_n = Dc_n$.
2d	:	Total waiting time at each workstation (W_n). Waiting happens if a certain operation cannot be started due to the precedence relationship. In this case, waiting time per unit (w_n) happens if $c_n < c_{n-1}$. Total waiting time at workstation n is the waiting time per unit multiplied by $(D-1)$; $w_n = c_{n-1} - c_n$; $W_n = w_n(D-1)$.
2e	:	Calculate the completion time for all unit at workstation-n by adding start time, total cycle time and waiting time, $F_n = s_n + C_n + W_n$.
2f	:	State the capacity sufficiency status (SS) of workstation-n; $SS = \begin{cases} Sufficient, & if\ F_n \geq H \\ Insufficient, & otherwise \end{cases}$
2g	:	Calculate the idle time for each workstation: deduct the total operational time by the completion time for all unit; $I_n = H - F_n$
Step 3	:	Check the sufficiency status for each workstation. If all workstation's sufficiency status is "sufficient" then go to step 8, else, go to step 4.
Step 4	:	Calculate the total idle time for all workstations; $I_{tot} = \sum_{n=1}^{N} I_n$
Step 5	:	Determine if the problem is solvable through adjusting the idle time (sub algorithm-2): if $I_{tot} \geq 0$, then go to sub algorithm 2, else go to step 6.
Step 6	:	Calculate the total waiting time for all workstations; $W_{tot} = \sum_{n=1}^{N} W_n$
Step 7	:	Determine if the problem is solvable through reducing waiting time (sub algorithm-3): if $I_{tot} + W_{tot} \geq 0$, then go to sub algorithm 3, else go to step 9.
Step 8	:	Declare that there is no line balancing problem. Go to step 10.
Step 9	:	Declare that the problem is unsolvable trough workload reallocation. Go to step 10.
Step 10	:	Terminate the algorithm.
Output	:	Workstations profile

TABLE II
SUB ALGORITHM 2 (TASK REALLOCATION THROUGH IDLE TIME ADJUSTMENT)

Input	:	Workstations profile from sub-algorithm 1
Step 1	:	Compile list of insufficient workstations along with their positions on the assembly line (see step 2f on sub-algorithm 1); $WS_n^{insuf} = \{W_n\}$ where $I_n < 0$. Arrange the list of insufficient workstations having the most upstream position to downstream. Begin with the first workstation on the list.
Step 2	:	Calculate the amount of processing time which needs to be reallocated (R_n^t) from a particular insufficient workstation. Convert the processing time into units (R_n^u); $R_n^t = \lvert I_n \rvert$; $R_n^u = \frac{\lvert I_n \rvert}{c_n}$.
Step 3	:	Create list of possible helpers for the insufficient workstations. The helper should have $I_n > 0$ and positioned before the insufficient workstations on the assembly line (upstream). $WS_n^{help} = \{W_n\}$ with $I_n > 0, n < WS_n^{insuf}$. Arrange the list of helpers having the most upstream position to downstream. Begin with the first workstation on the list.
Step 4	:	Calculate the capability of amount of available idle time which can be allocated to help the insufficient workstations. Convert the idle time into units. $H_n^t = I_n$; $H_n^u = \frac{I_n}{c_n}$.
Step 5	:	Reallocate work from insufficient workstation (identified on step 2) to the helper according to their capability (identified on step 4). Identify the helper's completion time associated with their previous work and use it as their start time as duplicates for the insufficient workstations. Eliminate WS_n^{help} from the list of helpers.
Step 6	:	Subtract the amount of reallocated work at WS_n^{insuf} from their load, recalculate their completion time and check whether insufficiency is resolved. If it is not, go back to step 3, else erase the workstation from the list of insufficient workstations and go to step 7.
Step 7	:	Check whether all insufficient workstation on the list is helped. If yes go to step 8, if not go back step 2.
Step 8	:	Terminate the algorithm.
Output	:	New task allocation at workstations

TABLE III
SUB ALGORITHM 3 (TASK REALLOCATION THROUGH IDLE AND WAITING TIME ADJUSTMENT)

Input	:	Workstations profile from sub-algorithm 1
Step 1	:	Compile list of insufficient workstations along with their positions on the assembly line (see step 2f on sub-algorithm 1). $WS_n^{insuf} = \{W_n\}$ with $I_n < 0$.
Step 2	:	Start with the most downstream workstation on the assembly line. Determine its latest completion time (based on working hours). Determine its latest start by moving backwards incorporating the total cycle time at the workstation. Set the latest start as the starting time of the most downstream workstation
Step 3	:	Identify the most upstream workstation. Use the available time from the most downstream process to help the most upstream workstation.
Step 4	:	Calculate the effect of duplicating the most upstream workstation toward each downstream workstation. If insufficiency is resolved, then go to step 6, else go to step 5.
Step 5	:	Renew the profile of insufficient workstations and check whether another adjustment is needed. If yes, go back to step 2, else go to step 6.
Step 6	:	Terminate the algorithm.
Output	:	New task allocation at workstations

II. Algorithm Implementation

An assembly line which consists of 3 workstations (WS_1, WS_2 and WS_3) is designed based on real world situation at a particular manufacturing company. Each workstation is assigned to a single undivided operation. The precedence relationship is serial, which means that operation-1 is the predecessor of operation-2 and operation-2 is the predecessor of operation-3.

TABLE IV
LIST OF OPERATIONS, PRECEDENCE RELATIONSHIP AND WORKSTATION ASSIGNMENT

Operation #	Predecessor for	Currently Assigned to
Operation-1	Operation-2	Workstation-1
Operation-2	Operation-3	Workstation-2
Operation-3	-	Workstation-3

The proposed algorithm will be implemented to four cases (Case 1, 2, 3 and 4) as listed on Table V. In Case 1, the situation was set where the operation time at $WS_1 < WS_2 < WS_3$, which means that the bottleneck workstation is positioned at the end of the assembly line. Case 2 represents a situation where the operation time at $WS_2 < WS_1 < WS_3$; in this case the bottleneck workstation is still at the end of the line (as in Case 1) but with different conditions of the feeder workstations. In Case 3, the operation time at $WS_3 < WS_2 < WS_1$, which means that the bottleneck workstation is at the very beginning of the

assembly line, slowing down all other downstream processes. Finally, in the last case (Case 4), the situation is set where the operation time at $WS_3 < WS_1 < WS_2$, which means that the bottleneck workstation is at the middle of the assembly line, therefore it is being a receiver as well as feeder towards another workstation on the assembly line. These four cases resemble situations commonly found on real world assembly lines where bottleneck workstations act as feeder, receiver or both.

TABLE V
ASSEMBLY LINE TEST CASES PROFILE

Case/ Operation time at Workstation	Operation time per unit at WS_n (in seconds)		
	WS_1	WS_2	WS_3
Case 1	27	40	44
Case 2	40	27	44
Case 3	44	40	27
Case 4	40	44	27

The demand for the system is 695 units per day and this daily demand should be fulfilled through 8 working hours (therefore the takt-time is 41,44 seconds). Each workstation processes one unit product at a time. After a particular unit is finished, it will be sent to the next workstation as their WIP. The time to move between workstations is neglected.

Case 1 - The situation was set where the operation time at $WS_1 < WS_2 < WS_3$. This means that the bottleneck workstation is positioned at the end of the assembly line. Executing step 1 to 3 on sub-algorithm 1 into this case would result in the profile shown in Table VI. WS_3 is listed as insufficient workstation (marked by negative idle time), whereas WS_1 and WS_2 as sufficient workstations (marked by positive idle time).

TABLE VI
CASE 1: RESULT AFTER IMPLEMENTING SUB-ALGORITHM 1

WS_n	s_n (in seconds)	c_n (in seconds)	C_n (in seconds)	W_n (in seconds)	F_n (in seconds)	Capacity Sufficiency	I_n (in seconds)
WS_1	0	27	18,765	-	18,765	Sufficient	10,035
WS_2	27	40	27,800	-	27,827	Sufficient	973
WS_3	67	44	30,580	-	30,647	Insufficient	-1,847

Further execution of step 4 and 5 sub-algorithm 1 leads to the calculation of positive total idle time in the system which is 9,161 seconds. This means that to eliminate insufficiency, task reallocation should be done by implementing sub-algorithm 2. The result is shown in Table VII.

TABLE VII
CASE 1: RESULT AFTER IMPLEMENTING SUB-ALGORITHM 2

WS_n	s_n (in seconds)	c_n (in seconds)	C_n (in seconds)	W_n (in seconds)	F_n (in seconds)
WS_1	0	27	18,765	-	18,765
WS_2	27	40	27,800	-	27,827
WS_3	67	44	28,732	-	28,799
WS_3 (by WS_1)	18,765	44	1,848	-	20,613

Given that WS1 is the first workstation listed as helper for WS3, soon after WS1 completed its predetermined tasks it must function as helper for WS3. It starts at 18,765 seconds and ends at 20.613 seconds (see the last line on Table VII). As a helper, WS1 helps WS3 processing 44 units of products for 1,848 seconds. Following this, the load of WS3 decreases by 44 units, leading to faster completion time (28,799 seconds). The new completion time is still within the working period limit; thus the insufficiency problem at WS3 is solved.

Case 2 – The case represents a situation where the operation time at $WS_2 < WS_1 < WS_3$; the bottleneck workstation is still at the end of the line (as in Case 1) but with different conditions of the feeder workstations. Table VIII shows the result of executing Step 1 to 3 on Sub-algorithm 1 into this case.

TABLE VIII
CASE 2: RESULT AFTER IMPLEMENTING SUB-ALGORITHM 1

WS_n	s_n (in seconds)	c_n (in seconds)	C_n (in seconds)	W_n (in seconds)	F_n (in seconds)	Capacity Sufficiency	I_n (in seconds)
WS_1	0	40	27,800	-	27,800	Sufficient	1,000
WS_2	40	27	18,765	9,022	27,827	Sufficient	973
WS_3	67	44	30,580	-	30,647	Insufficient	-1,847

Table VIII shows that WS3 suffers inefficiency (negative idle time) while WS1 and WS2 do not (positive idle time). Further execution of step 4 and 5 of Sub-algorithm 1 suggest that to eliminate insufficiency at WS3, Sub-algorithm 2 should be implemented. The result can be seen on Table IX.

TABLE IX
CASE 2 RESULT AFTER IMPLEMENTING SUB-ALGORITHM 2

WS_n	s_n (in seconds)	c_n (in seconds)	C_n (in seconds)	W_n (in seconds)	F_n (in seconds)
WS_1	0	40	27,800	-	27,800
WS_2	40	27	18,765	9,022	27,827
WS_3	67	44	28,644		28,711
WS_3 (by WS_1)	27,800	44	968		28,768
WS_3 (by WS_2)	27,827	44	968		28,795

Table VIII shows that both WS1 and WS2 are functioning as helpers for WS3. This is necessary given that the idle time at WS1 could not cover the total insufficiency of WS3, therefore WS2 is utilized. Both workstations start functioning as helpers for WS3 right after completing their predetermined tasks. Each workstation helps WS3 processing 22 units of products for 968 seconds. Following this, the load of WS3 decreases by 44 units, leading to faster completion time (28,711 seconds). The new completion time is still within the working period limit; thus, the insufficiency problem at WS3 is solved.

Case 3 – In Case 3, the operation time at $WS_3 < WS_2 < WS_1$. It means that the bottleneck workstation is at the most upstream position, slowing down all other downstream processes. Table X shows the result of executing Step 1 to 3 on Sub-algorithm 1 into this case.

TABLE X
CASE 3: RESULT AFTER IMPLEMENTING SUB-ALGORITHM 1

WS_n	s_n (in seconds)	c_n (in seconds)	C_n (in seconds)	W_n (in seconds)	F_n (in seconds)	Capacity Sufficiency	I_n (in seconds)
WS_1	0	44	30,580	-	30,580	Insufficient	-1,780
WS_2	44	40	27,800	2,776	30,620	Insufficient	-1,820
WS_3	84	27	18,765	11,798	30,647	Insufficient	-1,847

As mentioned before, WS_1 slows down all other downstream processes and causes WS_2 and WS_3 to have insufficient capacity yet high level of waiting time. All workstations cannot finish their predetermined task on time. Execution of step 4 and 5 on Sub-algorithm 1 will suggest the implementation of Sub-algorithm 3 for task reallocation, which means that task reallocation is done by adjusting the total waiting time. The result can be seen on Table IX.

TABLE XI
CASE 3: RESULT AFTER IMPLEMENTING SUB-ALGORITHM 3

WS_n	s_n (in seconds)	c_n (in seconds)	C_n (in seconds)	W_n (in seconds)	F_n (in seconds)
WS_1	0	44	20,548	-	20,548
WS_1 (by WS_3)	3	44	10,032	-	10,035
WS_2	44	40	27,800	-	27,844
WS_3	10,035	27	18,765	-	28,800

Sub-algorithm 3 works backward, beginning from WS_3 as the most downstream workstation. Waiting time for WS_3 is high, yet it cannot finish its tasks within the time limit, so the focus is shifting this waiting time into productive time by functioning WS_3 as helper for WS_1. The latest finish of WS_3 for its own tasks is at 28,800 seconds; the amount of time needed by WS_3 to process 695 units of products is 18,765 seconds, therefore its latest start is at 10,035 seconds. Using this information, the latest finish of WS_3 helping WS1 is at 10,035 seconds. This amount of time can be used to process 228 units of product for WS1. Following this the load of WS1 decreases by 228 units, leading to faster completion time (20,548 seconds). The new completion time is still within the working period limit; thus, the insufficiency problem at WS1 is solved. Solving insufficiency problems at WS1 speeds up WS_2 on processing its tasks. Waiting time for WS_2 is eliminated, and therefore the completion time then fall within the time limit. Insufficiency problems at all workstations are solved.

Case 4 – In this last case, the situation is set where the operation time at $WS_3 < WS_1 < WS_2$, which means that the bottleneck workstation is at the middle of the assembly line. Therefore, the bottleneck is being a receiver as well as feeder towards another workstation on the line. Table XII shows the result of executing step 1 to 3 on Sub-algorithm 1 into this case.

TABLE XII
CASE 4: RESULT AFTER IMPLEMENTING SUB-ALGORITHM 1

WS_n	s_n (in seconds)	c_n (in seconds)	C_n (in seconds)	W_n (in seconds)	F_n (in seconds)	Capacity Sufficiency	I_n (in seconds)
WS_1	0	40	27,800	-	27,800	Sufficient	1,000
WS_2	40	44	30,580	-	30,620	Insufficient	-1,820
WS_3	84	27	18,765	11,798	30,647	Insufficient	-1,847

Table XII shows that the total idle time for the system in Case 4 is negative, but the total idle and waiting time is positive. Based on this profile, step 4 and 5 on Sub-algorithm 1 will suggest the implementation of Sub-algorithm 3 for task reallocation. The result can be seen on Table XIII.

TABLE XIII
CASE 4: RESULT AFTER IMPLEMENTING SUB-ALGORITHM 3

WS_n	s_n (in seconds)	c_n (in seconds)	C_n (in seconds)	W_n (in seconds)	F_n (in seconds)
WS_1	0	40	17,800	-	17,800
WS_1 (by WS_3)	35	40	10,000		10,035
WS_2	40	44	19,580	-	19,620
WS_2 (by WS_1)	17,800	44	11,000	-	28,800
WS_3	10,035	27	18,765	-	28,765

Starting from WS_3 as the most downstream workstation. The latest finish of WS_3 for its own tasks is at 28,800 seconds; the amount of time needed by WS_3 to process 695 units of products is 18,765 seconds, therefore its latest start is at 10,035 seconds. At this point, WS_3 can straightly be functioned as helper for WS_2 (the bottleneck workstation). But helping WS_2 without considering their feeder (WS_1) would create waiting time at WS_2. Thus, step 2 on Sub Algorithm 2 suggests that WS_3 should be functioned as helper for WS_1 as the most upstream workstation. The latest finish of WS_3 helping WS1 is at 10,035 seconds. This amount of time can be used to process 250 units of product for WS1. Following this, the load of WS1 decreases by 250 units, leading to faster completion time (17,800 seconds). The new completion time is still within the working period limit. It also creates idle time for WS_1 which can now be used to help WS_2. The starting time for WS_1 helping WS_2 is at 17,800 seconds and able to process 250 units of product for WS_2. The load of WS_2 is then decrease by 250 units and its completion time is shortened to 19,620 seconds. Insufficiency problems at all workstations are solved.

RESULT AND DISCUSSION

The numerical experiment explained in the previous section shows that the proposed algorithm managed to reallocate tasks to match demand level. Table XIV shows adjustment towards workstation cycle time before (c_n) and after (c'_n) adjustment is made.

TABLE XIV
CYCLE TIME ADJUSTMENT AFTER TASKS REALLOCATION

Case #	WS_n	c_n (in seconds)	Adjustment (in seconds)	c'_n (in seconds)
Case 1	WS_1	27	2.66	29.66
	WS_2	40	0.00	40.00
	WS_3	44	-2.66	41.34
Case 2	WS_1	40	1.39	41.39
	WS_2	27	1.39	28.39
	WS_3	44	-2.79	41.21
Case 3	WS_1	44	-14.43	29.57
	WS_2	40	0.00	40.00
	WS_3	27	0.00	27.00
Case 4	WS_1	40	1.44	41.44
	WS_2	44	-15.83	28.17
	WS_3	27	14.39	41.39

Given the profile on Table VII, Table IX, Table XI, Table XIII and Table XIV, the system performance before and after tasks reallocation can be summarized as in Table XV.

TABLE XV
ASSEMBLY LINE PERFORMANCE BEFORE AND AFTER TASKS REALLOCATION

Performance Indicator	Case 1		Case 2		Case 3		Case 4	
	Before	After	Before	After	Before	After	Before	After
Demand fulfillment	No	Yes	No	Yes	No	Yes	No	Yes
Takt-time achievement	No	Yes	No	Yes	No	Yes	No	Yes
Smoothing Index	61.85	11.76	51.78	13.00	48.26	16.67	51.78	13.27

Table XV shows that demand fulfillment and takt time achievement is 100% after the algorithm is implemented and all cases show improvement on the line smoothing index, which shows improvement regarding load balance between workstations. Experiment on larger case (15 workstations) shows similar patterns, which proves the effectiveness of the proposed algorithm.

CONCLUSION AND RECOMMENDATION

The results presented in this paper highlight the capability of the proposed algorithm to dynamically reallocate tasks among workstations in response to demand fluctuations in manual-heavy reliance assembly line, where parallel processing could only be achieved by utilizing existing workers. The algorithm systematically identifies (1) which workstations experience capacity insufficiencies, (2) which workstations possess the capability to assist by duplicating the work of the insufficient stations, and (3) the optimal quantity of tasks to be reallocated from the insufficient workstations to the assisting stations. It is capable of functioning effectively across a variety of real-world assembly line scenarios, regardless of whether the bottleneck occurs upstream, downstream, or in the middle of the production line. Implementation of the algorithm ensures fulfillment of demand, adherence to the targeted takt-time, and improvement in overall line efficiency.

Further research should focus on comparing the performance of the proposed algorithm with alternative approaches that allow the addition of extra workers to the system as a means of enabling parallel processing. Such a comparison would provide decision-makers with a more comprehensive understanding of the trade-offs between operational performance and the investment required to implement each option

REFERENCES

[1] A. Qattawi and S. Chalil Madathil, "Assembly line design using a hybrid approach of lean manufacturing and balancing models," *Prod Manuf Res*, vol. 7, no. 1, pp. 125–142, Jan. 2019, doi: 10.1080/21693277.2019.1604274.

[2] S. Tanjong Tuan, M. Radzi, B. Haji, and C. Daud, "Optimizing Lean Manufacturing Efficiency with Novel Line Balancing in the Automotive Exhaust Manufacturing Sector," 2023.

[3] V. Sosa-Perez, J. Palomino-Moya, C. Leon-Chavarri, C. Raymundo-Ibañez, and F. Dominguez, "Lean Manufacturing Production Management Model focused on Worker Empowerment aimed at increasing Production Efficiency in the textile sector," in *IOP Conference Series: Materials Science and Engineering*, Institute of Physics Publishing, Apr. 2020. doi: 10.1088/1757-899X/796/1/012024.

[4] E. Álvarez-Miranda, S. Chace, and J. Pereira, "Assembly line balancing with parallel workstations," *Int J Prod Res*, vol. 59, no. 21, pp. 6486–6506, 2021, doi: 10.1080/00207543.2020.1818000.

[5] H. Fazlollahtabar, H. Hajmohammadi, and A. Es'Haghzadeh, "A heuristic methodology for assembly line balancing considering stochastic time and validity testing," *International Journal of Advanced Manufacturing Technology*, vol. 52, no. 1–4, pp. 311–320, Jan. 2011, doi: 10.1007/s00170-010-2708-1.

[6] H. F. Lee and R. V. Johnson, "A Line-Balancing Strategy for Designing Flexible Assembly Systems," Kluwer Academic Publishers, 1991.

[7] P. P. Pal, V. Karandikar, S. Kuber, and V. Desai, "Efficiency Improvement and Capacity Enhancement using Line Balancing Technique," *International Journal of Current Engineering and Technology*, vol. 10, no. 04, pp. 531–535, Jul. 2020, doi: 10.14741/ijcet/v.10.4.6.

[8] A. Syaharani Ibnu and A. U. Khasanah, "Enhancing Line Efficiency Performance at Assembly Line Using ECRS-Based Line Balancing Concept."

[9] A. Kumar and R. Kumar, "Productivity Improvement of Assembly Line-by-Line Balancing Technique: Case Study Textile Manufacturing Company Karachi Pakistan," *South Asian Journal of Social Studies and Economics*, vol. 21, no. 5, pp. 54–72, Mar. 2024, doi: 10.9734/sajsse/2024/v21i5814.

[10] W. Zhang, L. Hou, and R. J. Jiao, "Dynamic takt time decisions for paced assembly lines balancing and sequencing considering highly mixed-model production: An improved artificial bee colony optimization approach," *Comput Ind Eng*, vol. 161, p. 107616, Nov. 2021, doi: 10.1016/J.CIE.2021.107616.

[11] G. N. Nur, M. A. Sadat, and B. M. Shahriar, "Assembly line balancing considering stochastic task times and production defects."

[12] T. C. Lopes, A. S. Michels, C. G. S. Sikora, N. Brauner, and L. Magatão, "Assembly line balancing for two cycle times: Anticipating demand fluctuations," *Comput Ind Eng*, vol. 162, p. 107685, Dec. 2021, doi: 10.1016/J.CIE.2021.107685.

[13] N. P. Campana, M. Iori, and M. C. O. Moreira, "Exact and heuristic solutions for the assembly line balancing problem with hierarchical worker assignment," Apr. 2020, [Online]. Available: http://arxiv.org/abs/2004.13396

[14] A. S. Michels and A. M. Costa, "Model and heuristics for the multi-manned assembly line worker integration and balancing problem," *Int J Prod Res*, vol. 62, no. 24, pp. 8719–8744, 2024, doi: 10.1080/00207543.2024.2347572.

[15] J. Miltenburg, "U-shaped production lines: A review of theory and practice," *Int J Prod Econ*, vol. 70, no. 3, pp. 201–214, Apr. 2001, doi: 10.1016/S0925-5273(00)00064-5.

[16] I. Kucukkoc and D. Z. Zhang, "Mixed-model parallel two-sided assembly line balancing problem: A flexible agent-based ant colony optimization approach," *Comput Ind Eng*, vol. 97, pp. 58–72, Jul. 2016, doi: 10.1016/J.CIE.2016.04.001.

[17] I. Kucukkoc, "Balancing of two-sided disassembly lines: Problem definition, MILP model and genetic algorithm approach," *Comput Oper Res*, vol. 124, p. 105064, Dec. 2020, doi: 10.1016/J.COR.2020.105064.